

This Page Is Inserted by IFW Operations
and is not a part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images may include (but are not limited to):

- BLACK BORDERS
- TEXT CUT OFF AT TOP, BOTTOM OR SIDES
- FADED TEXT
- ILLEGIBLE TEXT
- SKEWED/SLANTED IMAGES
- COLORED PHOTOS
- BLACK OR VERY BLACK AND WHITE DARK PHOTOS
- GRAY SCALE DOCUMENTS

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images,
please do not report the images to the
Image Problem Mailbox.**

#find the quant. level by shifting by q, for the inverse it comes from the Huffman decoder#

LET

lev_data = BARREL_SHIFT_RIGHT(q,lev_final).

#saturate the lev at 37, for the Huffman table, except in lpf still mode, send all the bits#

lev_forw = CASE mode

OF lpf still:lev_data

ELSE CASE lev_data GT_U b'00000100101"

OF: b'00000100101"

ELSE lev_data

ESAC

ESAC,

lev = MUX_2(STRING(input_exp+1)bit){

lev_forw,

b'0" CONC lev_inv,

dir_sel).

#the level = 0 flag#

lev_z = lev EQ_U ZERO(input_exp+1)b'0".

inv_lev_z = CASE lev_z

OF tb'0

ELSE b'1

ESAC,

#the level value shifted up, and rounded#

round_lev = BARREL_SHIFT_LEFT(q,lev) AND_B

CASE mode

OF lpf still: b'00" CONC ALL_SAME(input_exp-1)b'1"

ELSE BIT_STRING(input_exp+1)(input_exp+1)inv_lev_z) ## lev==0 out all 0's#

```

        ESAC,
        #clear out extra bit for lpf still case#

        #calculate the proposed value: in the case n-o_ round_ lev is unsigned 10 bit, so result needs 11 bits#
        #pro_no will always be in range as round_ lev < (n-o) #

        pro_no = ADD SUB_ST(odd_ round_ lev, CASE sgn_ level
            OF b'0': add,
               b'1': sub;
            ESAC);

        #now pro_new = +/- round_ lev#

        round_sel = CASE sgn_ level
            OF b'0': left,
               b'1': right;
            ESAC;

        pro_new = MUX_2(STRING(input_exp+1)bit)(
            round_ lev,
            (NEG_U round_ lev)[2..input_exp+2], #NEG sign extends#
            round_sel);

        out_sel = CASE difference
            OF diff: left,
               ELSE right;
            ESAC;

        OUTPUT (MUX_2(STRING(input_exp)bit)(

```

```

pro_no[3..input_exp+2],
pro_new[2..input_exp+1],
out_sel),
lev[2..input_exp+1],
sgn_level)

```

```

END.
#actel 1 bit full adder with active low cin and cout#
FN FA1B = (bit: a1n b1n c1nb) -> (bit: bit1, bit1): #c1nb, s#
BEGIN
LET a_c = B_TO_S a1n CONC NOT_B(B_TO_S c1nb),
b_c = B_TO_S b1n CONC NOT_B(B_TO_S c1nb),
out = ADD_U(a_c, b_c),
OUTPUT(CAST(bit1) NOT_B(B_TO_S out[1]), out[2])
END.

```

#a Ripple carry adder using 1 bit full adder blocks#

#the actel version of the ADD BIOP's#

```

MAC ADD_S_ACTEL = (STRING(INT m) bit: a1n, STRING(INT n) bit: b1n, bit: c1nb) -> STRING(IF m>=n THEN m+1 ELSE n+1) FI bit:
BEGIN
MAKE (IF m>=n THEN m ELSE n FI) FA1B: sum.

```

#signed nos so sign extend #

```

LET a_c = IF m>=n THEN a1n ELSE ALL_SAME(n-m) B_TO_S a1n[1] CONC a1n FI,
b_c = IF n>=m THEN b1n ELSE ALL_SAME(m-n) B_TO_S b1n[1] CONC b1n FI.
LET subsignal = sum.
#lsb#

```

```

JOIN  (a_c||f m>=n THEN m ELSE n f),b_c||f m>=n THEN m ELSE n f),c||b) ->sum||f m>=n THEN m ELSE n f).

FOR INT j=1..(IF m>=n THEN m ELSE n f) - 1
JOIN (a_c||f m>=n THEN m ELSE n f) - j),b_c||f m>=n THEN m ELSE n f) - j),
sum||f m>=n THEN m ELSE n f) - j+1||1) ->sum||f m>=n THEN m ELSE n f) - j).

OUTPUT CAST(STRING||f m>=n THEN m+1 ELSE n+1 f)||bit)
(NOT_B(B TO S sum||1||1) CONC
CAST(STRING||f m>=n THEN m ELSE n f)||bit)( (INT j=1..IF m>=n THEN m ELSE n f) sum||j||2))
END.

MAC ADD_US_ACTEL = (STRING(INT m)||bit:a||n,STRING(INT n)||bit:b||n,bit:c||b) ->STRING||f m>=n THEN m+1 ELSE n+1 f)||bit:
BEGIN
MAKE ||f m>=n THEN m ELSE n f)||FA1B:sum.

#unsigned nos so extend by 0#
LET a_c = IF m>=n THEN a||n ELSE ZERO(n-m)||b'0' CONC a||n f,
b_c = IF n>=m THEN b||n ELSE ZERO(m-n)||b'0' CONC b||n f.
LET subsignal = sum.

#lsb#
JOIN (a_c||f m>=n THEN m ELSE n f),b_c||f m>=n THEN m ELSE n f),c||b) ->sum||f m>=n THEN m ELSE n f).

FOR INT j=1..(IF m>=n THEN m ELSE n f) - 1
JOIN (a_c||f m>=n THEN m ELSE n f) - j),b_c||f m>=n THEN m ELSE n f) - j),
sum||f m>=n THEN m ELSE n f) - j+1||1) ->sum||f m>=n THEN m ELSE n f) - j).

OUTPUT CAST(STRING||f m>=n THEN m+1 ELSE n+1 f)||bit)
(NOT_B(B TO S sum||1||1) CONC
CAST(STRING||f m>=n THEN m ELSE n f)||bit)( (INT j=1..IF m>=n THEN m ELSE n f) sum||j||2)) )

```

```

END.

MAC_ADD_SUB_ST = (STRING(INT m)bit:ain, STRING(INT n)bit:bin, !_add:sel) -> STRING(IF m>=n THEN m+1 ELSE n+1 FI)bit;

BEGIN

#sign extend inputs#
LET a_s = CAST(STRING(1)bit:ain[1]) CONC ain,
    b_s = CAST(STRING(1)bit:bin[1]) CONC bin,
    sel_bit = CAST(STRING(1)bit:sel,
#ACTEL#
    bin_inv = XOR_B(n+1)(b_s, ALL_SAME(n+1)sel_bit),

#cnb is active low so cast sel(add->0,sub->1) & invert it#
    out = ADD_S_ACTEL(a_s, bin_inv, CAST(bit:NOT_B sel_bit),
    binout = out[2..IF m>=n THEN m+2 ELSE n+2 FI]

OUTPUT binout
END.

#transformation ops#
MAC_B_TO_S = (bit:in) -> STRING(1)bit: CASE in
    OF b'0:b'0",
        b'1:b'1"
    ESAC.

MAC_I_TO_SC(INT n) = (t_result:in) -> (flag, STRING(n)bit): BIOP_TRANSFORM_S.
MAC_SC_TO_I(INT n) = (STRING(n)bit:in) -> (flag, result): BIOP_TRANSFORM_S.

MAC_S_TO_IN = (STRING(INT n)bit:in) -> (flag, input): BIOP_TRANSFORM_S.
MAC_IN_TO_S(INT n) = (input:in) -> (flag, STRING(n)bit): BIOP_TRANSFORM_S.

```

```

MAC U_TO_IN = (STRING(INT n)bit:in) -> (flag, l_input): BIOP TRANSFORM_US.

MAC U_TO_LEN = (STRING(INT n)bit:in) -> (flag, l_length): BIOP TRANSFORM_US.
MAC_LEN_TO_U(INT n) = (l_length:in) -> (flag, STRING(n)bit): BIOP TRANSFORM_US.

MAC Q_TO_U(INT n) = (l_quant:in) -> (flag, STRING(n)bit): BIOP TRANSFORM_US.
MAC S_TO_C = (STRING(INT n)bit:in) -> (flag, l_col): BIOP TRANSFORM_US.
MAC S_TO_R = (STRING(INT n)bit:in) -> (flag, l_row): BIOP TRANSFORM_US.
MAC S_TO_B = (STRING(INT n)bit:in) -> (flag, l_blk): BIOP TRANSFORM_US.
MAC S_TO_SUB = (STRING(INT n)bit:in) -> (flag, l_sub): BIOP TRANSFORM_US.
MAC S_TO_SPARC = (STRING(INT n)bit:in) -> (flag, l_sparc_addr): BIOP TRANSFORM_US.

MAC C_TO_S(INT n) = (l_col:in) -> (flag, STRING(n)bit): BIOP TRANSFORM_US.
MAC R_TO_S(INT n) = (l_row:in) -> (flag, STRING(n)bit): BIOP TRANSFORM_US.

MAC I_TO_Q = (l_input:in) -> l_quant: ARITH in.

```

```

MAC B_TO_I = (bit:in) -> l_result: CASE in
  OF b'0: result 0,
     b'1: result 1
  ESAC.

```

```

MAC CARRY = (l_add:in) -> STRING(1)bit: CASE in
  OF add: b'0",
     sub: b'1"
  ESAC.

```

```

MAC BOOL_BIT = (boot:in) -> STRING(1)bit:
CASE in
OF b'1"

```

ELSE b'0'
ESAC.

MAC BOOL_STRING(INT n) = (n)bool:in) ->STRING(n) bit:
(LET out = BOOL_BIT ln[1].
OUTPUT IF n=1
THEN out
ELSE out[1] CONC BOOL_STRING(n-1)(n[2..n])
FI
).

MAC BIT_STRING(INT n) = (n)bit:in) ->STRING(n) bit:
(LET out = B_TO_S ln[1].
OUTPUT IF n=1
THEN out
ELSE out[1] CONC BIT_STRING(n-1)(n[2..n])
FI
).

MAC ZERO(INT n) = (STRING[1]bit:dummy) ->STRING(n)bit:
IF n=1 THEN b'0'
ELSE b'0' CONC ZERO(n-1) b'0'
FI.

MAC ALL_SAME(INT n) = (STRING[1]bit:dummy) ->STRING(n)bit:
IF n=1 THEN dummy
ELSE dummy CONC ALL_SAME(n-1) dummy
FI.

COM

The operators described in this section are optimal and take two-valued operands and produce a two-valued result. They may not be used with ELLA-integers or associated types.

The first basic value of any two-valued type declaration of the operand(s) and the result are interpreted by the operations as false, and the second basic value is interpreted as true. Thus, given the following type declarations:

MOC

MAC AND_T = (TYPE t a b) -> t: BIOP AND.

MAC OR_T = (TYPE t: a b) -> t: BIOP OR.

MAC XOR_T = (TYPE t: a b) -> t: BIOP XOR.

MAC NOT_T = (TYPE t: a) -> t: BIOP NOT.

COM

The following operations take bit-string operand(s) and are bitwise, i.e. the operation is performed on the operand(s) one bit at a time. The operand(s) and result must all be ELLA-strings of the same length.

MOC

MAC AND_B = (STRING(int) nbit, STRING(nbit)) -> STRING(nbit):
BIOP AND.

MAC OR_B = (STRING(int) nbit, STRING(nbit)) -> STRING(nbit):
BIOP OR.

MAC XOR_B = (STRING[INT n]bit, STRING[n]bit) -> STRING[n]bit;
BIOP XOR.

MAC NOT_B = (STRING[INT n]bit) -> STRING[n]bit;
BIOP NOT.

COM

The operators described in this section may be used with primitive types ie all enumerated types, except associated types, rows, strings and structures. These operations take two operands which must be of the same type and the result can be any two-valued type; we have packaged these BIOPs so they output a value of type 'bool' - you may change this if you wish.

MOC

MAC EQ = (TYPE t: a b) -> bool: BIOP EQ.

MAC GT = (TYPE t: a b) -> bool: BIOP GT.

MAC GE = (TYPE t: a b) -> bool: BIOP GE.

MAC LT = (TYPE t: a b) -> bool: BIOP LT.

MAC LE = (TYPE t: a b) -> bool: BIOP LE.

COM

NOTE: these BIOPs are designed to take any primitive ELLA type. Since it is not possible to distinguish between primitive and other types, whilst leaving the macro declaration general enough to allow the use of all two-valued types that might be declared, there are type-checking limitations. This is done at network assembly, so use of illegal types will not generate an error

message until then.

NB: ARITH provides for relational operations on ELLA integer types.

MOC

COM

These operations are optimal in their handling of '?' and operate on bit-string representations of unsigned integers. The result may be any two-valued type; we have used type 'bool'. The inputs can be of different lengths and different types.

MOC

MAC EQ_U = (STRING[INT n]b4, STRING[INT m]b4) -> bool:
BIOP EQ_US.

MAC GT_U = (STRING[INT n]b4, STRING[INT m]b4) -> bool:
BIOP GT_US.

MAC GE_U = (STRING[INT n]b4, STRING[INT m]b4) -> bool:
BIOP GE_US.

MAC LT_U = (STRING[INT n]b4, STRING[INT m]b4) -> bool:
BIOP LT_US.

MAC LE_U = (STRING[INT n]b4, STRING[INT m]b4) -> bool:
BIOP LE_US.

Bit-strings representing signed numbers

COM

These operations are optimal and operate on bit-string representations of signed integers. The result may be any two-valued type; we have used type

'bool'. The inputs can be of different lengths and different types.

MOC

MAC EQ_S = (STRING[INT n]bit, STRING[INT m]bit) -> bool:
BIOP EQ_S.

MAC GT_S = (STRING[INT n]bit, STRING[INT m]bit) -> bool:
BIOP GT_S.

MAC GE_S = (STRING[INT n]bit, STRING[INT m]bit) -> bool:
BIOP GE_S.

MAC LT_S = (STRING[INT n]bit, STRING[INT m]bit) -> bool:
BIOP LT_S.

MAC LE_S = (STRING[INT n]bit, STRING[INT m]bit) -> bool:
BIOP LE_S.

Shift operations

COM

These operate on bit-strings. Both the enclosing macro and the BIOP are parameterised by the number of bits to be shifted (INT p). The macro and BIOP parameters must match. Note that no bits are lost in these shift operations, so you may need to trim the result to achieve the desired effect.

SR means shift right; SL means shift left.

The macros with the suffix '_S' perform arithmetic shifts; those with the

suffix 'U' perform bool shifts.
MOC

MAC SL_S(INT p) = (STRING(INT n)bit) -> STRING(n + p)bit:
BIOP SL(p).

MAC SL_U(INT p) = (STRING(INT n)bit) -> STRING(n + p)bit:
BIOP SL(p).

MAC SR_S(INT p) = (STRING(INT n)bit) -> STRING(n + p)bit:
BIOP SR_S(p).

MAC SR_U(INT p) = (STRING(INT n)bit) -> STRING(n + p)bit:
BIOP SR_US(p).

Arithmetic operations

Bit-strings representing unsigned numbers

addition.

MAC ADD_U = (STRING(INT m)bit, STRING(INT n)bit)
-> STRING(IF m >= n THEN m+1 ELSE n+1 F)bit:
BIOP PLUS_US.

subtraction on bit-string representations of unsigned integers. Output is #
signed.

MAC SUB_U = (STRING(INT m)bit, STRING(INT n)bit)
-> STRING(IF m >= n THEN m+1 ELSE n+1 F)bit:

BIOP MINUS_US.

negation. Output is signed.

MAC NEG_U = (STRING(INT n)bit) -> STRING(n+1)bit;
BIOP NEGATE_US.

multiplication.

MAC MULT_U = (STRING(INT m)bit, STRING(INT n)bit) -> STRING(m+n)bit;
BIOP TIMES_US.

COM

- divide. If the divisor is non-zero then the first element of the output is 'ok' and the second and third elements are the quotient and remainder; otherwise, the first element is 'error' and the rest is set to '7'.

MOC

MAC DIV_U = (STRING(INT m)bit, STRING(INT n)bit)
-> (flag, STRING(m)bit, STRING(n)bit):

BIOP DIVIDE_US.

square root.

MAC SORT_U = (STRING(INT n)bit) -> STRING((n+1) % 2)bit;
BIOP SORT_US.

COM

modulus (result always positive). If the divisor is non-zero, then the first element of the output is 'ok' and the second element is the modulus; otherwise, the first element is 'error' and the second is '7'.

MOC

MAC MOD_U = (STRING(INT m)bit, STRING(INT n)bit)
 -> (flag, STRING(n)bit):
 BIOP MOD_US.

COM

- convert between one range of bit-string and another. If the input value cannot be represented as a legal value for the output string, the result is 'error' and '?'.
 MOC

MAC RANGE_U (INT m) = (STRING(INT n)bit)
 -> (flag, STRING(m)bit):
 BIOP RANGE_US.

Bit-strings representing signed numbers

addition.

MAC ADD_S = (STRING(INT m)bit, STRING(INT n)bit)
 -> STRING(IF m >= n THEN m+1 ELSE n+1 F)bit:
 BIOP PLUS_S.

subtraction.

MAC SUB_S = (STRING(INT m)bit, STRING(INT n)bit)
 -> STRING(IF m >= n THEN m+1 ELSE n+1 F)bit:
 BIOP MINUS_S.

negation.

MAC NEG_S = (STRING(INT n)bit) -> STRING(n+1)bit;
 BIOP NEGATE_S.

multiplication.

MAC MULT_S = (STRING(INT m)bit, STRING(INT n)bit) -> STRING(m+n)bit;
 BIOP TIMES_S.

COM

divide. If the divisor is non-zero then the first element of the output is 'ok' and the second and third elements are the quotient and remainder; otherwise, the first element is 'error' and the rest is set to '?'. The remainder has the same sign as the divisor.

MOC

MAC DIV_S = (STRING(INT m)bit, STRING(INT n)bit)
 -> (flag, STRING(m)bit, STRING(n)bit);
 BIOP DIVIDE_S.

COM

modulus (result always positive). If the divisor is non-zero, then the first element of the output is 'ok' and the second element is the unsigned modulus; otherwise, the first element is 'error' and the second is '?'.
 MOC

MOC

MAC MOD_S = (STRING(INT m)bit, STRING(INT n)bit)
 -> (flag, STRING(n)bit);
 BIOP MOD_S.

COM

- convert between one range of bit-string and another. If the input value cannot be represented as a legal value for the output string, the result is 'error' and '?'.
MOC

MAC RANGE_S (INT m) = (STRING(INT n)bit)

-> (flag, STRING(m)bit):

BIOP RANGE_S.

absolute value. The output represents an unsigned integer.

MAC ABS_S = (STRING(INT n)bit) -> STRING(n)bit:

BIOP ABS_S.

Built in Register

MAC DREG(INT interval delay) = (TYPE t) -> t:

ALIEN REGISTER (interval, ?t, 0, delay).

MAC GEN_DREG(INT interval, CONST (TYPE t): init, INT skew delay) = (t) -> t:

ALIEN REGISTER (interval, init, skew, delay).

Built in type conversion

MAC CAST(TYPE t) = (TYPE s) -> t:

ALIEN CAST.

```

MAC ALL_SAME(INT n) = (STRING(1)bit:dummy) ->STRING(n)bit:
BEGIN
  FAULT IF n < 1 THEN "N<1 in ALL_SAME" FI.
  OUTPUT IF n=1 THEN dummy
  ELSE dummy CONC ALL_SAME(n-1) dummy
  FI
END.

MAC CAST (TYPE to) = (TYPE from) ->to:ALIEN CAST.

MAC ZERO(INT n) = (STRING(1)bit:dummy) ->STRING(n)bit:
BEGIN
  FAULT IF n < 1 THEN "N<1 in ZERO" FI.
  OUTPUT IF n=1 THEN b"0"
    ELSE b"0" CONC ZERO(n-1) b"0"
    FI
  END.

MAC B_TO_S= (bit:in) ->STRING(1)bit: CASE in
  OF b"0:b"0",
    b"1:b"1"
  ESAC.

MAC S_TO_IN = (STRING(input_exp)bit:in) -> (flag,1,input): BIOP TRANSFORM S.
MAC IN_TO_S(INT n) = (1,input:in) -> (flag,STRING(n)bit): BIOP TRANSFORM S.

MAC S_HUFF = (STRING(s)bit) -> (flag,1,huffman): BIOP TRANSFORM US.
MAC HUFF_S = (1,huffman) -> (flag,STRING(s)bit): BIOP TRANSFORM US.

MAC BOOL_BIT = (bool:in) ->STRING(1) bit:

```

```

CASE in
OF !:b'1'
ELSE b'0'
ESAC.
MAC BIT_BOOL = (bit:in)    ->bool:
CASE in
OF b'1'
ELSE !
ESAC.

```

```

MAC BOOL_STRING(INT n) = ([n]:bool:in) ->STRING[n] bit:
(LET out = BOOL_BIT in[1].
OUTPUT IF n=1
THEN out
ELSE out[1] CONC BOOL_STRING(n-1) ([n]:[2..n])
FI

```

```

).
# defines the types used for the 2D wavelet chip#

```

```

#constant values#
INT  result_exp=14,    #length of result arith#
     input_exp=10,     #length of 1D convolver input/output#
     qmax = 7,         #maximum shift value for quantisation constant#
     result_range = 1 SL (result_exp-1),
     input_range = 1 SL (input_exp-1),
     max_octave=3, #no of octaves=max_octave+1, can not be less in this example#
     no_octave=max_octave+1, #""#
     xsize = 10, #no of bits for ximage#
     ysize = 9, #no of bits for yimage#
     ximage=319, #the xdimension -1 of the image, ie no of cols#

```

yimage=239 #the ydimension -1 of the image, ie no of rows#

```
#int types#
TYPE t_result= NEW result/( -(result_range)..(result_range-1)).
t_input= NEW input/( -(input_range)..(input_range-1)).
t_length= NEW len/(0..15).
t_inp = NEW inp/(0..1023).
t_bik =NEW blk/(0..3).
t_sub =NEW sub/(0..3).
t_col =NEW col/(0..ximage).
t_row =NEW row/(0..yimage).
t_carry =NEW carry/(0..1).
t_quant =NEW quant/(0..qmax).
#address for result&dwf memory, ie 1 frame#
t_sparc_addr =NEW addr/(0..(1 SL max_octave)*(ximage+1)*(yimage+1)+(pimage+1))-1).
t_octave=NEW ocl/(0..(max_octave+1)).
```

#bit string and boolean types#

```
bit = NEW b(0 | '1).
bool = NEW (f|f).
flag = NEW(error | ok).
```

#control signals#

```
t_reset = NEW(rst|no_rst).
t_load = NEW(write|read), #r/wbar control#
t_cs = NEW(no_select|select), #chip select control#
t_updown= NEW(down|up), #up/down counter control#
t_diff= NEW(diff|nodiff), #diff or not in quantiser#
t_intra = NEW(intra|inter).
```

```

#convolver mux & and types#
  t_mux = NEW(left|right),
  t_mux3 = NEW(|c|r),
  t_mux4 = NEW(unr|dos|res|quatro),
  t_add = NEW(add|subtl),
  t_direction=NEW(forward|inverse),

#counter types#
  t_count_control=NEW(count_rsl|count_carry),
  t_count_2 = NEW(one|two),

#state types#
  t_token = NEW (l_0|l_1|l_11|l_100|l_101),
  t_mode= NEW(void|void_still|stop|send|still|still_send|lpf_send|lpf_still|lpf_stop),
  t_cycle = NEW(token_cycle|data_cycle|skip_cycle),
  t_state= NEW(start|up|up1|zz0|zz1|zz2|zz3|down1),
  t_decode = NEW(load_low|load_high),
  t_high_low = NEW(low|high),
  t_huffman = NEW(pass|huffman),
  t_fifo = NEW(nk_fifo|error_fifo),
  #types for the octave control unit#
  t_channel= NEW(y|u|v),
  t_channel_factor= NEW(luminance|color),
  #types for the control of memory ports#
  t_sparcport=(l_sparc_addr#wr_addr#l_sparc_addr#rd_addr#l_load#wr#l_csl|csl)

#generate random values for test memories#

FN GEN_RANDOM_MEM = (bool:ck,l_reset:reset) ->l_input: BOOL_INT10 PRBS11(ck,reset).
TYPE t_test = NEW(no|yes).
#.....#
#These functions change types from boolean to integer and vice-#

```

```

#versa. Supports 1 & 8 bit booleans.
#.....#

FN INT_BOOL1=(l_input:k) ->bool: # 1bit input to binary #
CASE k
OF input/0:1,
input/1:1
ESAC.

FN BOOL_INT=(bool:b) ->l_input: # 1 bit bool to input #
CASE b
OF:input/0,
input/1
ESAC.

FN * =(l_input:a b) ->l_input: ARITH a*b.
FN % =(l_input:a b) ->l_input: ARITH a%b.
FN - =(l_input:a b) ->l_input: ARITH a-b.
FN + =(l_input:a b) ->l_input: ARITH a+b.
FN = =(l_input:a b) ->l_test: ARITH IF a=b THEN 2 ELSE 1 FI.

COM
FN CHANGE_SIGN =(l_input:i) ->l_input: #changes sign for 8-bit 2's#
ARITH IF i<0 THEN 128+i #complement no, #
ELSE i
FI.

FN SIGN =(l_input:i) ->bool: #gets sign for 2's#
ARITH IF i<0 THEN 1 #complement nos #
ELSE 2

```

FI.

```

FN TEST_SIZE = (l_input:x) -> bool:
#tests to see if the input is bigger than an 8-bit integer#
ARITH IF ( (x<=-128) AND (x>127)) THEN 1
      ELSE 2 FI.

```

```

FN INT8_BOOL=(l_input:orig) -> [8]bool:
BEGIN

```

```

SEQ
VAR i1:=input/0, #input variables#
i0:=CHANGE_SIGN(orig),
b:=[1,1,1,1],SIGN(orig));
[INT n=1..7] (
  i1:=i0%input/2;
  b[n]:=INT_BOOL1(i0-input/2**i1);
  i0:=i1
);
OUTPUT CASE TEST_SIZE orig #checks to see if orig will#
OF: [8]bool, #fit input to an 8 bit value#
f, b
ESAC
END.

```

```

FN BOOL_INT8=( [8]bool:b) -> l_input: #converts 8bit boolean to 2's#
BEGIN
SEQ
#complement integer #
VAR sum:=input/-128 * BOOL_INT([8]).
exp:=input/1;

```

```

[INT k=1..7]
( sum:=sum+exp*BOOL_INT(b[k]);
  exp:=input/2 * exp
);
OUTPUT sum
END.

```

```

MOC
FN BOOL_INT10=(l[10]bool:b) ->l_input: #converts 10bit boolean to 2's#
BEGIN
  SEQ #complement integer #
  VAR sum:=input/-512 * BOOL_INT(b[10]).
  exp:=input/1;
  [INT k=1..9]
  ( sum:=sum+exp*BOOL_INT(b[k]);
    exp:=input/2 * exp
  );
  OUTPUT sum
END.
COM
FN BOOL_INT16=(l[16]bool:in1 in2) ->l_input:
# converts a 16-bit no., (lsbs,msbs) into integer form#
(BOOL_INT8(in1))+((input/256)*BOOL_INT8(in2))+((input/256)*BOOL_INT8(in1[6])).
#hack because of sign extend#
#of lsb #
MOC
COM
FN PRBS10 = (l_reset:reset) ->[10]bool:
#A 10 bit prbs generator, feedback taps on regs 3 & 10.#
BEGIN

```



```
MAKE[10]MYLATCH:I,
XNOR:xnor.
```

```
FOR INT k=1..9 JOIN
(reset,[k]) ->[k+1].
```

```
JOIN (reset,xnor) ->[1],
([10],[3]) ->xnor.
```

```
OUTPUT 1
```

```
END.
```

```
MOC
```

```
FN PRBS11 = (bool:ck,i reset:reset) ->[10]bool:
```

```
#A 11 bit prbs generator,feedback taps on regs 2 & 11.#
```

```
BEGIN
```

```
MAKE[11]DIFF[bool]:i,
```

```
XOR:xor.
```

```
FOR INT k=1..10 JOIN
```

```
(ck,reset,[k],i) ->[k+1].
```

```
JOIN (ck,reset,NOTxor,i) ->[1],
([1],i[2]) ->xor.
```

```
OUTPUT [1..10]
```

```
END.
```

```
COM
```

```
FN PRBS16 = (bool:reset)->[16]bool:
```

```
#A 16 bit prbs generator,feedback taps on regs 1,3,12,16#
```

```
BEGIN
```

```
MAKE[16]MYLATCH:I,
```

```
XOR_4xor,
NOTxor.
```

```
FOR INT k=1..15 JOIN
  (ck,reset,[k])->[k+1].
```

```
JOIN (ck,reset,xnor) ->[1],
      ([1],[3],[16],[12]) ->xor,
xor ->xnor.
```

```
OUTPUT ([INT k=1..16][k])
```

```
END.
```

```
FN PRBS12 = (clock:ck,boot:reset) ->[12]boot:
#A 12 bit prbs generator,feedback taps on regs 1,4,6,12.#
```

```
BEGIN
```

```
  MAKE[12]MYLATCH1,
```

```
  XOR_4xor,
```

```
  NOTxor.
```

```
FOR INT k=1..11 JOIN
  (ck,reset,[k])->[k+1].
```

```
JOIN (ck,reset,xnor) ->[1],
      ([1],[4],[6],[12]) ->xor,
xor ->xnor.
```

```
OUTPUT ([INT k=1..12][k])
```

```
END.
```

```
FN PRBS8 = (clock:ck,boot:reset) ->[8]boot:
#A 8 bit prbs generator,feedback taps on regs 2,3,4,8.#
```

```
BEGIN
```

```
  MAKE[8]MYLATCH1,
```

```
XOR_4:xor,
NOT:xnor.
```

```
FOR INT k=1..7 JOIN
  (ck,reset,[k])->[k+1].
```

```
JOIN (ck,reset,xnor) ->[1],
      ([2],[3],[4],[8]) ->xor,
xor ->xnor.
```

```
OUTPUT ([INT k=1..8])[k]
END.
```

```
MOC
```

```
#test for palmas chip#
```

```
TYPE l_int32 = NEW int32/(-2147483000..2147483000).
```

```
FN RMS = (bool:ck,l_reset:reset,l_cycle:cycle,t_input:old new) ->l_int32:
BEGIN
```

```
FN I_32 = (l_input:in) ->l_int32:ARITH in.
FN DV = (l_int32:a b) ->l_int32:ARITH a%b.
FN PL = (l_int32:a b) ->l_int32:ARITH a+b.
FN MI = (l_int32:a b) ->l_int32:ARITH a-b.
FN TI = (l_int32:a b) ->l_int32:ARITH a*b.
```

```
MAKEDFF_INIT(l_int32):old_error.
```

```
LET err = l_32old MI l_32new,
err2 = (err1err) PL old_error.
```

```
JOIN (ck,reset,CASE cycle
      OF data_cycle:write
```

```

ELSE read
  ESAC,ent2,int32/0)    ->old_error.

OUTPUT old_error
END.

FNEQ = (!_input:a b) ->bool:ARITH IF a=b THEN 2
      ELSE 1
      FI.

FN SPARC_MEM = (!_input:in,t_sparc_addr:wr_addr,t_sparc_addr:rd_addr,t_load:rw_sparc,t_cs:cs#)->t_input:
RAM(input/0).

FN FIFO = (bool:ck,t_reset:reset,STRING(16)bit:buffer_in,t_direction:direction,t_load:filo_read_filo_write)
->(STRING(16)bit,(2)t_filo): #filo_full,empty#
BEGIN
  FN FIFO_RAM = (STRING(16)bit:in,t_inp:wr_addr rd_addr,t_load:rw_filo) ->STRING(16)bit:
RAM(b'00000000000000000000').

  FN FULL = (!_inp:in) ->!_filo:ARITH IF in>1023 THEN 2 #filo full#
            ELSE 1
            FI.

  FN INCR = (!_inp:in) ->!_inp:ARITH in+1.

  FN EMPTY = (!_inp:in) ->!_filo:ARITH IF in<0 THEN 2 #filo empty#
            ELSE 1
            FI.

  FN DECR = (!_inp:in) ->!_inp:ARITH in-1.

MAKEDFF(!_inp,address,

```

```

FIFO_RAM:ram.

LET next = CASE direction
OF forward: CASE fifo_write
  OF write: INCR address
  ELSE address
  ESAC,
  inverse: CASE fifo_read
  OF read: INCR address
  ELSE address
  ESAC
ESAC.

JOIN (ck, reset, next, inp0) -> address,
      (buffer_in, address, address, CASE direction
      OF inverse: read,
      forward: fifo_write
      ESAC) -> ram.

OUTPUT (ram, (FULL address, EMPTY address))
END.

FN TEST_PALMAS = (boot.ck, t_reset: reset, t_direction: direction, t_intra: intra_inter, t_channel_factor: channel_factor,
                  t_input: q_int, t_quant: quant_norm, t_result: threshold_comparison)
-> (STRING(16) bit, #buffer_out#[2] t_load#fifo_read fifo_write# boot, boot, t_int32):

BEGIN

```

MAKE SPARC_MEM:new_old_inv old_forw,
FIFO.fifo,
PALMAS:palmas_inv palmas_forw.

```

LET   col_length = (IN_TO_S(9) input/31)[2],
      row_length = (IN_TO_S(9) input/31)[2],
      ximage_string = (IN_TO_S(9) input/32)[2],
      yimage_string = (IN_TO_S(9) input/32)[2],
      yimage_string_3 = (IN_TO_S(9) input/80)[2],
      pro_forw = palmas_forw[1],
      pro_inv = palmas_inv[1],
      forw_frame_done = palmas_forw[7],
      inv_frame_done = palmas_inv[7],
      cycle = palmas_inv[8],
      old_equal = CASE cycle
                    OF data_cycle:old_forw EQ palmas_inv[1]
                    ELSE
                    ESAC.

```

JOIN

```

#fix fifo full/empty logic later#
(ck,reset,forward,intra_inter,channel_factor,q_int,quant_norm,b'0000000000000000',new_old_forw,threshold,comparison,
#fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo,col_length,row_length,ximage_string,yimage_string,yimage_string_3)
->palmas_forw,

(ck,reset,inverse,intra_inter,channel_factor,q_int,quant_norm,fifo[1],new_old_inv,threshold,comparison,
#fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo,col_length,row_length,ximage_string,yimage_string,yimage_string_3)
->palmas_inv,

#old forward mem, on forward use as normal, on inverse read values to compare with inverse#
(pro_forw,CASE direction
  OF forward:palmas_forw[2],
    inverse:palmas_inv[2]
  ESAC, CASE direction
  OF forward:palmas_forw[2],
    inverse:palmas_inv[2]
  ESAC,CASE direction
  OF forward:palmas_forw[4][1],
    inverse:read
  ESAC)    ->old_forw,

(palmas_inv[1],palmas_inv[2],palmas_inv[2],CASE direction
  OF forward:read,
    inverse:palmas_inv[4][1]
  ESAC)    ->old_inv,

#(input0,palmas_forw[2],palmas_forw[2],palmas_forw[3][1])    ->new,#
(input0,CASE direction
  OF forward:palmas_forw[2],
    inverse:palmas_inv[2]
  ESAC, CASE direction

```

- 631 -

```

OF forward:palmas_forw[2],
   inverse:palmas_inv[2]
ESAC,CASE direction
OF forward:palmas_forw[3][1],
   inverse:read
   ESAC)
   ->new,

```

```

(ck,reset,CASE direction
OF inverse:b"0000000000000000",
   forward:palmas_forw[5]
   ESAC
   ,direction, palmas_inv[6][1], palmas_forw[6][2]) ->fifo.

```

```

OUTPUT (palmas_forw[5], palmas_forw[6], palmas_forw[7], old_equal, RMS(ck, reset, cycle, old_inv, new) )
END.

```

```

#test for palmas chip#

```

```

TYPE t_int32 = NEW int32/(-2147483000..2147483000).

```

```

FN RMS = (bool:ck, t_reset:reset, t_cycle:cycle, t_input:old new) -> t_int32:
BEGIN

```

```

FN I_32 = (t_input:in) -> t_int32:ARITH in.
FN DV = (t_int32:a b) -> t_int32:ARITH a%b.
FN PL = (t_int32:a b) -> t_int32:ARITH a+b.
FN MI = (t_int32:a b) -> t_int32:ARITH a-b.
FN TI = (t_int32:a b) -> t_int32:ARITH a*b.

```

```

MAKE DFF_INIT(t_int32:old_error.
LET err = t_32old MI t_32new,
err2 = (errTIerr) PL old_error.

```



```

JOIN (ck,reset,CASE cycle
      OF data_cycle:write
      ELSE read
      ESAC,err2,in32/0) ->old_error.

OUTPUT old_error
END.
FNEQ = (!_input.a b) ->bool:ARITH IF a=b THEN 2
      ELSE 1
      FI.

FN SPARC_MEM = (!_input.in,!_sparc_addr.wr_addr,!_sparc_addr.rd_addr,!_load.rw_sparc,!_cs:cs#)->!_input:
RAM(input/0).

FN FIFO_BIG = (bool:ck,!_reset:reset,STRING[16]bit:buffer.in,!_direction:direction,!_load.fifo_read fifo_write)
->(STRING[16]bit,[2]_fifo): #fifo_full,empty#

BEGIN
  FN FIFO_RAM = (STRING[16]bit:in,!_sparc_addr.wr_addr rd_addr,!_load.rw_fifo) ->STRING[16]bit:
RAM(b"00000000000000000000").

  FN FULL = (!_sparc_addr.in) ->!_fifo:ARITH IF in>1023 THEN 2 #fifo_full#
      ELSE 1
      FI.

  FN INCR = (!_sparc_addr.in) ->!_sparc_addr:ARITH in+1.

  FN EMPTY = (!_sparc_addr.in) ->!_fifo:ARITH IF in<0 THEN 2 #fifo_empty#
      ELSE 1
      FI.

  FN DECR = (!_sparc_addr.in) ->!_sparc_addr:ARITH in-1.

```

```

MAKE DFF(i_spare_addr):address,
  FIFO_RAM:ram.
LET next = CASE direction
  OF forward: CASE fifo_write
    OF write:NCR address
    ELSE address
    ESAC,
    Inverse: CASE fifo_read
    OF read:NCR address
    ELSE address
    ESAC
  ESAC.

```

```

JOIN (ck,reset,next,addr/0) ->address,
  (buffer_in,address,address,CASE direction
    OF inverse:read,
      forward:fifo_write
    ESAC) ->ram.

```

```

OUTPUT (ram,(FULL address,EMPTY address))
END.

```

```

FN TEST_PALMAS = (bod:ck,t_reset:reset, bootload_memory,t_direction:direction,t_intra:intra_inter,
  t_channel_factor:channel_factor,(4)t_quant:quant_norm,(4)t_result:threshold,
  t_input: col_length_in row_length_in ximage_string_in yimage_string_in,
  t_result:yimage_string_3_in)

```

```

->(bod#1, int32#1):

```

```

BEGIN
  FN NEW_ADDRESS = (!_sparc_addr:in)      -> !_sparc_addr: ARITH ((in + 1) MOD 120000).

  MAKE SPARC_MEM: new old_inv old_forw,
  FIFO BIG: fifo,
  PRBS11: prbs,
  OFF(!_sparc_addr): address,
  PALMAS: palmas.

  LET   col_length = (IN_TO_S(10) col_length_in)[2],
        row_length = (IN_TO_S(9) row_length_in)[2],
        ximage_string = (IN_TO_S(10) ximage_string_in)[2],
        yimage_string = (IN_TO_S(9) yimage_string_in)[2],
        yimage_string_3 = (!_TO_SC(11) yimage_string_3_in)[2],
        pro = palmas[1],
        random_data = BOOL_INT10 prbs,
        frame_done = palmas[7],
        cycle = palmas[8],
        old_equal = CASE cycle

```

- 635 -

```

OF data_cycle:old_forw EQ palmas[1]
ELSE1
ESAC.

JOIN
#fix fifo full/empty logic later#
(ck,reset,direction,intra_inter,channel_factor,quant_norm,CASE direction
OF forward:b"0000000000000000"
ELSE fifo[1]
ESAC,new,CASE direction
OF forward:old_forw
ELSE old_inv
ESAC,threshold,
#fifo[2][1],fifo[2][2]#ok_fifo,ok_fifo,col_length,row_length,ximage_string,yimage_string_3)
->palmas,
(ck,reset,(NEW_ADDRESS address),addr0) -> address,

(ck,reset) ->prbs,

#old forward mem, on forward use as normal, on inverse read values to compare with inverse#
(CASE load_memory
OF t:DFRq_input)(ck,reset,random_data,input0)
ELSE palmas[1]
ESAC , CASE load_memory
OF t:address
ELSE palmas[2]
ESAC, palmas[2], CASE load_memory
OF t:write
ELSE CASE direction
OF forward:palmas[4][1],

```

```

inverse:read
ESAC
ESAC)    ->old_forw,

(CASE load_memory
OF t:DIFF(t_input){ck,reset,random_data,input(0)
ELSE palmas[1]
ESAC , CASE load_memory
OF t:address
ELSE palmas[2]
ESAC, palmas[2], CASE load_memory
OF t:write
ELSE CASE direction
OF forward:read,
inverse:palmas[4][1]
ESAC
ESAC)    ->old_inv,

(CASE load_memory
OF t:random_data
ELSE input(0)
ESAC , CASE load_memory
OF t:address
ELSE palmas[2]
ESAC, palmas[2], CASE load_memory
OF t:write
ELSE CASE direction
OF forward:palmas[3][1],
inverse:read
ESAC
ESAC)    ->new,

```

```

(ck,reset,CASE direction
  OF  inverse:b"00000000000000000000".
    forward:palmas[5]
  ESAC  ,direction, palmas[6][1],palmas[6][2])  ->fifo.

OUTPUT (old_equal#,RMS(ck,reset,cycle,old_inv,new)# )
END.

#test for palmas chip#
TYPE l_int32 = NEW int32/(-2147483000..2147483000).

FN RMS = (bool:ck,l_reset:reset,l_cycle:cycle,l_input:old new)  ->l_int32
BEGIN

  FN l_32 = (l_input:in) ->l_int32:ARITH in.
  FN DV = (l_int32:a b) ->l_int32:ARITH a%b.
  FN PL = (l_int32:a b) ->l_int32:ARITH a+b.
  FN MI = (l_int32:a b) ->l_int32:ARITH a-b.
  FN TI = (l_int32:a b) ->l_int32:ARITH a*b.

  MAKEDFF_INIT(l_int32:old_error.

  LET err = l_32old MI l_32new,
    err2 = (errTIerr) PL old_error.

  JOIN (ck,reset,CASE cycle
    OF data_cycle:write
    ELSE read
    ESAC,err2,int32/0)  ->old_error.

```

OUTPUT odd_error
END.

FN EQ = (!_input:a b) -> bool:ARITH IF a=b THEN 2
ELSE 1
FI.

FN SPARC_MEM = (!_input:in, !_sparc_addr:wr_addr, !_sparc_addr:rd_addr, !_load:rw_sparc, !_cs:cs#) -> !_input:
RAM(inp:rd).

FN FIFO = (bool:ck, !_reset:reset, STRING[16]bit:buffer, !_in, !_direction:direction, !_load:ffo_read, !_write)
-> (!_STRING[16]bit[2](!_ffo): #ffo_full, empty#

BEGIN

FN FIFO_RAM = (STRING[16]bit:in, !_inp:wr_addr rd_addr, !_load:rw_ffo) -> STRING[16]bit:
RAM(b"00000000000000000000000000000000").

FN FULL = (!_inp:in) -> !_ffo:ARITH IF in>1023 THEN 2 #ffo_full#
ELSE 1

FI.

FN INCR = (!_inp:in) -> !_inp:ARITH in+1.

FN EMPTY = (!_inp:in) -> !_ffo:ARITH IF in<0 THEN 2 #ffo_empty#
ELSE 1

FI.

FN DECR = (!_inp:in) -> !_inp:ARITH in-1.

MAKEDFF(!_inp):address,
FIFO_RAM:ram.

```

LET      next = CASE direction
OF forward: CASE fifo_write
OF write: INCR address
ELSE address
ESAC,
inverse: CASE fifo_read
OF read: INCR address
ELSE address
ESAC
ESAC.

```

```
JOIN (ck,reset,next,lnp0)    ->address,
    (buffer_in,address,address,CASE direction
      OF inverse:read,
         forward:filo_write
      ESAC) ->ram.
```

```

OUTPUT      (ram,(FULL address,EMPTY address))
END.

```

```

FN TEST_PALMAS = (bool:clk,! reset:reset, bool:load_memory,! direction:direction,! intra_intra_inter,! channel_factor:channel_factor,
! input:q_int,! quant:quant_norm,! result:threshold comparison)

```

```

->(bool,1_int32):

```

BEGIN

```
FN NEW_ADDRESS = (l_sparc_addr:n)
->l_sparc_addr: ARITH ((n+1) MOD 120000).
```


- 640 -

```

MAKE SPARC_MEM:new old_inv old_forw,
FIFO:ffio,
PRBS11:prbs,
DFF(I_sparc_addr):address,
PALMAS:palmas.

LET    col_length = (IN_TO_S(10) input/31)[2],

row_length = (IN_TO_S(9) input/31)[2],

ximage_string = (IN_TO_S(10) input/32)[2],
yimage_string = (IN_TO_S(9) input/32)[2],
yimage_string_3 = (I_TO_SC(11) result/80)[2],

pro = palmas[1],

random_data = BOOL_INT10 prbs,

frame_done = palmas[7],

cycle = palmas[8],

old_equal = CASE cycle
    OF data_cycle:old_forw EQ palmas[1]
    ELSE 1
    ESAC.

```

```

JOIN
#fix fifo full/empty logic later#
(ck,reset,direction,intra_inter,channel_factor,q_int,quant_norm,fifo[1],new,CASE direction
  OF forward:old_forw
  ELSE old_inv
  ESAC, threshold,comparison,
  #fifo[2][1],fifo[2][2]#ok_fifo,ck_fifo,col_length,row_length,ximage_string,yimage_string_3)
  ->palmas,

(ck,reset,(NEW!_ADDRESS address),addr/0)      -> address,

(ck,reset)      ->prbs,

#old forward mem, on forward use as normal, on inverse read values to compare with Inverse#
(CASE load_memory
  OF t:DIFF(!_input){(ck,reset,random_data,input/0)
  ELSE palmas[1]}
  ESAC, CASE load_memory
  OF l:address
  ELSE palmas[2]
  ESAC, palmas[2], CASE load_memory
  OF l:write
  ELSE CASE direction
  OF forward:palmas[4][1],
  inverse:read
  ESAC
  ESAC)      ->old_forw,

(CASE load_memory
  OF t:DIFF(!_input){(ck,reset,random_data,input/0)
  ELSE palmas[1]}

```

```

ESAC , CASE load_memory
  OF t:address
  ELSE palmas[2]
  ESAC, palmas[2], CASE load_memory
    OF t:write
    ELSE CASE direction
      OF forward:palmas[4][1],
        inverse:read
      ESAC
    ESAC) ->old_inv,

```

```

(CASE load_memory
  OF t:random_data
  ELSE input/0
  ESAC , CASE load_memory
    OF t:address
    ELSE palmas[2]
    ESAC, palmas[2], CASE load_memory
      OF t:write
      ELSE CASE direction
        OF forward:palmas[3][1],
          inverse:read
        ESAC
      ESAC) ->new,

```

```

(ck,reset,CASE direction
  OF inverse:b'0000000000000000",
    forward:palmas[5]

```

ESAC ,direction,palmas[6][1],palmas[6][2]) ->ffo.
OUTPUT (old_equal,RMS(ct,reset,cycle,old_inv,new))
END.

- 644 -

APPENDIX C

7/22/93 3:39 PM

Engineering:KlicsCode:CompPict:Top.a

© Copyright 1993 KLICS Limited
All rights reserved.

Written by: Adrian Lewis

680X0 Fast Top Octave

seg 'klics'

macro

TOPX &DG, &HG, &old, &XX

```

swap      &HG          ; HG=G1H0
move.w    &DG,&XX       ; XX=G0
neg.w     &DG          ; DG=D(-G0)
add.w     &HG,&DG       ; DG=DD
add.w     &XX,&HG       ; HG=G1D
swap      &HG          ; HG=DG1
move.l    &DG,&old      ; save DD

```

endm

macro

TOPY &HG0, &new0, &HG1, &new1, &XX

```

move.l    &new0,&XX      ; read HG
move.l    &new1,&HG1     ; read HG
move.l    &HG1,&HG0      ; copy HG
add.l     &XX,&HG1       ; new1=H1G1
sub.l     &XX,&HG0       ; new0=H0G0

```

endm

macro

TOPBLOCK &DG0, &HG0, &new0, &old0, &DG1, &HG1, &new1, &old1, &XX

```

TOPY      &HG0,&new0,&HG1,&new1,&XX
TOPX      &DG0,&HG0,&old0,&XX
TOPX      &DG1,&HG1,&old1,&XX

```

endm

macro

TOPH &DG, &HG, &new, &old, &XX

```

move.l    &new,&HG
TOPX      &DG,&HG,&old,&XX

```

endm

macro

TOPE &DG, &old, &XX

```

move.l    &DG,&XX       ; XX=DG
swap      &XX           ; XX=GD
move.w    &XX,&DG       ; DG=DD
move.l    &DG,&old      ; save DD

```

- 646 -

Engineering:KlicsCode:CompPict:Top.e

```

      endm
-----
TopBwd  FUNC      EXPORT
PS      RECORD      8
src      DS.L        1
dst      DS.L        1
width    DS.L        1
height   DS.L        1
      ENDR

      link          a6,#0          ; no local variables
      movem.l       d4-d7/a3-a5,-(a7) ; store registers

      movea.l       PS.src(a6),a0   ; read src
      move.l        PS.height(a6),d7 ; read height
      move.l        PS.width(a6),d6 ; read width
      move.l        a0,a1           ; read dst
      move.l        PS.dst(a6),a1

      move.l        d6,d5           ; inc = width
      add.l         d5,d5           ; inc*=2
      move.l        d5,a4           ; save inc

      lsr.l         #1,d7           ; height/=2
      subq.l        #2,d7           ; height-=2

      lsr.l         #2,d6           ; width/=4
      subq.l        #2,d6           ; width-=2

      move.l        d6,d5           ; ccount=width
      move.l        (a0)+,d0        ; d0=new0++

@do1    TOPH         d0,d1,(a0)+,(a1)+,d4
      TOPH         d1,d0,(a0)+,(a1)+,d4
      dbf          d5,@do1          ; while -1!--ccount
      TOPH         d0,d1,(a0)+,(a1)+,d4
      TOPE         d1,(a1)+,d4

@do2    move.l      a0,a2           ; new0=new1
      move.l      a1,a3           ; old0=old1
      adda.l      a4,a0           ; new1+=inc
      adda.l      a4,a1           ; old1+=inc
      move.l      d6,d5           ; ccount=width
      TOPY        d2,(a2)+,d0,(a0)+,d4

@do3    TOPBLOCK    d2,d3,(a2)+,(a3)+,d0,d1,(a0)+,(a1)+,d4
      TOPBLOCK    d3,d2,(a2)+,(a3)+,d1,d0,(a0)+,(a1)+,d4
      dbf          d5,@do3          ; while -1!--ccount

      TOPBLOCK    d2,d3,(a2)+,(a3)+,d0,d1,(a0)+,(a1)+,d4
      TOPE        d1,(a1)+,d4
      TOPE        d3,(a3)+,d4
      dbf          d7,@do2          ; while -1!--height

      move.l      d6,d5           ; ccount=width
      add.l      #1,d5           ; d0=new0++

@do4    move.l      (a3)+,(a1)+     ; copy prev line
      move.l      (a3)+,(a1)+
      dbf          d5,@do4          ; while -1!--ccount

      movem.l     (a7)+,d4-d7/a3-a5 ; restore registers

```

- 647 -

Engineering:KlicsCode:CompPict:Top.a

```
unlk      a6      ; remove locals
rts       ; return
```

```
ENDFUNC
```

```
-----
END
```


- 648 -

Engineering:KlicsCode:CompPict:Table.a

```

-----
*
*  © Copyright 1993 KLICS Ltd.
*  All rights reserved.
*
-----

```

```

-----
*
*  680XC Table Lookup RGB/YUV code
*
-----

```

```

        machine    MC68030
        seg        'klics'

        if &TYPE('seg')*'UNDEFINED' then
        seg        &seg
        endif

MKTABLE FUNC      EXPORT
PS      RECORD      8
Table   DS.L        1
        ENDR

        link       a6,#0
        movem.l    d4-d7/a3-a5,-(a7)      ; store registers

        move.l     PS,Table(a6),a0        ; Table is (long)(2U*512) (long)(512-(6
        clr.l      d0                      ; U value

@MakeLoop
        move.w     #512,d1                ; 512

        move.l     d0,d2                    ; U
        move.w     d2,d3                    ; U

        add.w      d2,d2                    ; 2U
        add.w      d1,d2                    ; 2U + 512

        lsr.w      #2,d2                    ; 2U
        move.w     d2,(a0)+                  ; Place 1st word
        move.w     d2,(a0)+                  ; Place 2nd word

        add.w      d3,d3                    ; 2U
        move.w     d3,d2                    ; 2U
        add.w      d3,d3                    ; 4U
        add.w      d2,d3                    ; 6U
        asr.w      #4,d3                    ; 6U/16
        sub.w      d3,d1                    ; 512 - (6U/16)

        lsr.w      #2,d1                    ; 2U
        move.w     d1,(a0)+                  ; Place 1st word
        move.w     d1,(a0)+                  ; Place 2nd word

        add.w      #1,d0
        cmp.w      #50200,d0
        bne        @MakeLoop

        move.l     #500000200,d0            ; U value
        clr.l      d4

@MakeNegLoop
        move.w     #512,d1                ; 512

        move.w     d0,d2                    ; U

```

- 649 -

Engineering:KlicsCode:CompPic::Table.a

```

or.w      $FC00,d2
move.w    d2,d3          ;0

add.w     d2,d2          ;20
add.w     d1,d2          ;20 + 512

asr.w     #2,d2
move.w    d2,(a0)+       ;Place 1st word
move.w    d2,(a0)+       ;Place 2nd word
add.w     d3,d3          ;20
move.w    d3,d2          ;20
add.w     d3,d3          ;40
add.w     d2,d3          ;60
asr.w     #4,d3          ;60/16
sub.w     d3,d1          ;512 - (60/16)

asr.w     #2,d1
move.w    d1,(a0)+       ;Place 1st word
move.w    d1,(a0)+       ;Place 2nd word

add.l     #1,d0
add.l     #1,d4
cmp.w     #0200,d4
bne       @MakeNegLoop

movem.l   (a7)+,d4-d7/a3-a5 ; restore registers
unlk      a6              ; remove locals
rts       ; return

```

ENDFUNC

```

macro
FLXOV      &V, &SP1, &SP2

```

```

move.w    &V,&SP1
clr.b     &SP1
andi.w    #03FFF,&SP1
sne       &SP1
btst      #13,&SP1
seq       &SP2
or.b      &SP1,&V
and.w     &SP2,&V
swap     &V
move.w    &V,&SP1
clr.b     &SP1
andi.w    #03FFF,&SP1
sne       &SP1
btst      #13,&SP1
seq       &SP2
or.b      &SP1,&V
and.w     &SP2,&V
swap     &V

```

endm

```

if &TYPE('seg')!='UNDEFINED' then
seg      &seg
endif

```

```

YUV2RGB4   FUNC      EXPORT
PS         RECORD     8
Table      DS.L       1

```

- 650 -

Engineering:KlucsCode:CompPict:Table.a

```

pixmap DS.L    = 1
Y       DS.L    1
U       DS.L    1
V       DS.L    1
area    DS.L    1
width   DS.L    1
cols    DS.L    1
        ENDR

LS      RECORD      0,DECR
inc     DS.L        1
width   DS.L        1
fend    DS.L        1
count   DS.L        1
LSize   EQU        *
        ENDR

*void YUVtoRGB(Ptr TablePtr,long *pixmap,short *Yc,short *Uc,short *Vc,long area,1
*{
*long      inc,width,fend,count;
*      a0 = Y0, a1 = Y1, a2 = U, a3 = V, a4 = pm0, a5 = pm1
*      d0..6 = used, d7 = count

      link      a6,$LS.LSize      ; save locals
      movem.l    d0-d7/a0-a5,-(a7)  ; store registers

      move.l     PS.pixmap(a6),a4   ; pm0=pixmap
      move.l     a4,a5              ; pm1=pm0
      move.l     PS.Y(a6),a0        ; Y0=Yc
      move.l     a0,a1              ; Y1=Y0
      move.l     PS.U(a6),a2        ; U=Uc
      move.l     PS.V(a6),a3        ; V=Vc
      move.l     PS.area(a6),d7     ; fend=area
      lsl.l      #2,d7              ; fend<=2
      add.l      a4,d7              ; fend+=pm0
      move.l     d7,LS.fend(a6)     ; save fend
      move.l     PS.width(a6),d5    ; width=width
      move.l     d5,d7              ; count=width
      asr.l      #1,d7              ; count>>=1
      subq.l     #1,d7              ; count-=1
      move.l     d7,PS.width(a6)    ; save width

      add.l      d5,d5              ; width*=2
      add.l      d5,a1              ; Y1+=width
      add.l      d5,d5              ; width*=2
      move.l     d5,LS.width(a6)    ; save width

      move.l     PS.cols(a6),d4     ; inc=cols
      lsl.l      #2,d4              ; inc<=2
      add.l      d4,a5              ; pm1+=inc
      add.l      d4,d4              ; cols*=2
      sub.l      d5,d4              ; inc now 2*cols-widthn bytes
      move.l     d4,LS.inc(a6)      ; save inc

      move.l     a6,-(sp)
      move.l     PS.Table(a6),a6

; Colors wanted are:
;   RED    = (Y + 2V + 512) / 4
;   GREEN  = (Y - V + 512 - (60/16)) / 4
;   BLUE   = (Y + 2U + 512) / 4
;
; do      ; uv2rgb(*Y++,*V++,*U++)

```

UTable part is for (2V + 512)
 UTable part is for (512 - (60
 UTable part is for (2U + 512)

- 651 -

Engineering:KlicsCode:CompPict:Table.a

```

d1 - ra, d2 - ga, d3 - ba, d4 - rb, d5 - gb/512, d6 - bb

move.w    (a2)+,d2                ; U
beq       @DoQuickU
and.w     #03FF,d2
move.l    (a6,d2.w*8),d3          ;BLUE,Get (2U + 512)/4 for Blue = (Y +
move.l    d3,d6                  ;Dup for second pair
move.l    4(a6,d2.w*8),d5        ;GREEN, Get (512 - (6U/16))/4 for Gree
@DoQuickU
move.w    (a3)+,d1                ; V
beq       @DoQuickV              ;if zero then handle using the quick m
move.w    d1,d4
asr.w     #2,d1
sub.w     d1,d5                  ;GREEN, Get (512 - (6U/16) - V)/4 for
move.w    d5,d2
swap      d5
move.w    d2,d5
move.l    d5,d2                  ;Dup for second pair

and.w     #03FF,d4
move.l    (a6,d4.w*8),d4        ;RED, Get (2V + 512)/4 for Red = (Y +
move.l    d4,d1
bra       @TestEnd

@DoQuickU
move.l    #00800080,d3          ;BLUE,Get (2U + 512)/4 for Blue = (Y +
move.l    d3,d6                  ;Dup for second pair
move.l    d3,d5                  ;GREEN, Get (512 - (6U/16))/4 for Gree
bra       @DidQuickU

@DoQuickV
move.l    d5,d2                  ;GREEN, Get (512 - (6U/16) - V)/4 for
move.l    #00800080,d4          ;RED, Get (2V + 512)/4 for Red = (Y +
move.l    d4,d1                  ;Dup for second pair

@TestEnd

; add Ya to RGB values - FETCHY (a0)+,d0,d1,d2,d3
move.l    (a0)+,d0              ;Y
asr.w     #2,d0
swap      d0
asr.w     #2,d0
swap      d0                    ;Y is -128 to +127
add.l     d0,d1                  ;RED, Get (Y+ 2V + 512) for Red = (Y +
add.l     d0,d2                  ;GREEN, Get (Y + (512 - (6U/16)) - V)
add.l     d0,d3                  ;BLUE,Get (Y + (2U + 512) for Blue = (

; add Yb to RGB values - FETCHY2 (a1)+,d0,d4,d5,d6
move.l    (a1)+,d0              ;Y
asr.w     #2,d0
swap      d0
asr.w     #2,d0
swap      d0                    ;Y is -128 to +127
add.l     d0,d4                  ;RED, Get (Y+ 2V + 512) for Red = (Y +
add.l     d0,d5                  ;GREEN, Get (Y + (512 - (6U/16)) - V)
add.l     d0,d6                  ;BLUE,Get (Y + (2U + 512) for Blue = (

move.l    d1,d0
or.l      d4,d0

or.l      d2,d0
or.l      d3,d0
or.l      d5,d0

```

- 652 -

Engineering:KlicsCode:CompFict:Table.a

```

or.l    >    d6,d0

and.l    *$FF00FF00,d0
bre      @over                ; if overflow

@ok      ; save RGBa - MKRGB d1,d2,d3,(a4)-
ls1.l    #8,d2                ; G=G0G0 (12)
or.l     d3,d2                ; G=GBGB (12)
move.l   d1,d3                ; B=0R0R (12)
swap     d3                   ; B=0R0R (21)
move.w   d2,d3                ; B=0RGB (2)
swap     d2                   ; G=GBGB (21)
move.w   d2,d1                ; R=0RGB (1)
move.l   d1,(a4)+             ; *RGB++=rgb (1)
move.l   d3,(a4)+             ; *RGB++=rgb (2)

; save RGBb - MKRGB d4,d5,d6,(a5)-
ls1.l    #8,d5                ; G=G0G0 (12)
or.l     d6,d5                ; G=GBGB (12)
move.l   d4,d6                ; B=0R0R (12)
swap     d6                   ; B=0R0R (21)
move.w   d5,d6                ; B=0RGB (2)
swap     d5                   ; G=GBGB (21)
move.w   d5,d4                ; R=0RGB (1)
move.l   d4,(a5)+             ; *RGB++=rgb (1)
move.l   d6,(a5)+             ; *RGB++=rgb (2)

dbf      d7,@do                ; while

move.l   (sp)+,a6

adda.l   LS.inc(a6),a4         ; pm0+=inc
adda.l   LS.inc(a6),a5         ; pm1+=inc
adda.l   LS.width(a6),a0       ; Y0+=width
exg.l    a0,a1                 ; Y1<->Y0
move.l   PS.width(a6),d7       ; count=width
cmpa.l   LS.fend(a6),a4        ; pm0<fend
blt.w    @do2                  ; while

movem.l   (a7)+,d0-d7/a0-a5    ; restore registers
unlk     a6                    ; remove locals
rts

@do2     move.l   a6,-(sp)
move.l   PS.Table(a6),a6
bra      @do                    ; return

@FixIt   btst     #31,d0        ; See if upper word went negative
beq      @D1TopNotNeg
and.l    #50000FFFF,d0        ; Pin at zero
@D1TopNotNeg
btst     #24,d0                ; See if upper word went too positive
beq      @D1TopNotPos
and.l    #50000FFFF,d0        ; Mask old data out
or.l     #500FF0000,d0        ; New data is maxed
@D1TopNotPos

btst     #15,d0                ; See if lower word went negative
beq      @D1BotNotNeg

```

- 653 -

Engineering:KlicsCode:CompPict:Table.a

```

    and.l    *SFFFF0000,d0      :Pin at zero
@DlBotNotNeg >
    btest   #8,d0              :See if lower word went too positive
    beq     @DlBotNotPos
    and.l    *SFFFF0000,d0      :Mask old data out
    or.l     *50000000FF,d0     :New data is maxed
@DlBotNotPos
    rts

Gover
    move.l   d1,d0
    bsr     @FixIt
    move.l   d0,d1

    move.l   d2,d0
    bsr     @FixIt
    move.l   d0,d2

    move.l   d3,d0
    bsr     @FixIt
    move.l   d0,d3

    move.l   d4,d0
    bsr     @FixIt
    move.l   d0,d4

    move.l   d5,d0
    bsr     @FixIt
    move.l   d0,d5

    move.l   d6,d0
    bsr     @FixIt
    move.l   d0,d6

    bra     @ok

-----
ENDFUNC
END

```

Engineering:KlicsCode:CompPict:KlicsUtil.a

```

-----
*
*   © Copyright 1993 KLICS Limited
*   All rights reserved.
*
*   Written by: Adrian Lewis
*
-----
*
*   68000 Klics Utilities
*
-----
        seg      'klics'

KLCopy  FUNC      EXPORT
*
*   KLCOPY(short *src, short *dst, int area):
*
PS      RECORD      8
src      DS.L        1
dst      DS.L        1
end      DS.L        1
        ENDR

*
*   link          a6,#0                ; no local variables
*
*   move.l        PS.src(a6),a0        ; short *src
*   move.l        PS.dst(a6),a1        ; short *dst
*   move.l        PS.end(a6),d3        ; long area
*   lsr.l         #4,d3                ; in words(x8)
*   subq.l        #1,d3                ; area-=1
*do      move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         move.l        (a0)+,(a1)+      ; *dst++=*src++
*         dbf          d3,edo            ; if --area goto do
*
*   unlk          a6                    ; remove locals
*   rts                                ; return
*
        ENDFUNC
-----
KLHalf  FUNC      EXPORT
*
*   KLHALF(short *src, short *dst, long width, long height):
*   Dimensions of dst (width, height) are half that of src
*
PS      RECORD      8
src      DS.L        1
dst      DS.L        1
width    DS.L        1
height   DS.L        1
        ENDR

*
*   link          a6,#0                ; no local variables
*   movem.l        d4,-(a7)            ; store registers
*
*   move.l        PS.src(a6),a0        ; short *src
*   move.l        PS.dst(a6),a1        ; short *dst

```

- 655 -

Engineering:KlicsCode:CcmpPict:KlicsUtil.a

```

        move.l    _PS.width(a6),d2      ; long width
        move.l    PS.height(a6),d3     ; long height
        subq.l    #1,d3                 ; height-=1
@do_y    move.l    d2,d4                 ; count=width
        lsr.l     #2,d4                 ; count /= 2
        subq.l    #1,d4                 ; count-=1
@do_x    move.l    (a0)+,d0              ; d0=*src++
        move.w    (a0)+,d0              ; d2=*src++
        addq.l    #2,a0                 ; src+=1 short
        move.l    d0,(a1)+             ; *dst++=d0
        move.l    (a0)+,d0              ; d0=*src++
        move.w    (a0)+,d0              ; d2=*src++
        addq.l    #2,a0                 ; src+=1 short
        move.l    d0,(a1)+             ; *dst++=d0
        dbf       d4,@do_x             ; if -1!--width goto do_x
        adda.l    d2,a0                 ; skip a quarter row
        adda.l    d2,a0                 ; skip a quarter row
        adda.l    d2,a0                 ; skip a quarter row
        adda.l    d2,a0                 ; skip a quarter row
        dbf       d3,@do_y             ; if -1!--height goto do_y
        .
        movem.l   (a7)+,d4              ; restore registers
        unlk      a6                   ; remove locals
        rts                                     ; return
        .

```

ENDFUNC

KLZero FUNC EXPORT

* KLZERO(short *data, int area);

```

PS      RECORD      8
data    DS.L         1
end      DS.L         1
        ENDR

```

```

        link       a6,@0               ; no local variables
        .
        move.l     PS.data(a6),a0      ; short *data
        move.l     PS.end(a6),d3       ; long area
        lsr.l      #3,d3                ; in words(x4)
        subq.l     #1,d3                ; area-=1
@do      clr.l      (a0)+                ; *dst++=*src++
        clr.l      (a0)+                ; *dst++=*src++
        clr.l      (a0)+                ; *dst++=*src++
        clr.l      (a0)+                ; *dst++=*src++
        dbf        d3,@do               ; if -1!--area goto do
        .
        unlk       a6                   ; remove locals
        rts                                     ; return
        .

```

ENDFUNC

CLEARA2 FUNC EXPORT

```

        move.l     #0,a2
        rts
        .
        END

```


- 656 -

Engineering:KlicsCode:CompFict:KlicsEncode.h

```

.....
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
...../
typedef struct {
    int      bpf_in,      /* User - Bytes per frame in input stream */
            bpf_out,      /* User - Bytes per frame in output stream */
            buf_size;      /* User - Buffer size (bytes) */

    Boolean  intra,        /* Calc - Compression mode intra/inter */
            auto_q,        /* User - Automatic quantization for rate control */
            buf_sw;        /* User - Theoretical buffer on/off */

    float    quant,        /* User - Starting quantiser value */
            thresh,        /* User - Threshold factor */
            compare,        /* User - Comparison factor */
            base[5];        /* User - Octave weighting factors */

    int      buffer,        /* Calc - Current buffer fullness (bytes) */
            prevbytes,      /* Calc - Bytes sent last frame */
            prevquact;      /* Calc - Quantisation/activity for last frame */

    double   tmp_quant;    /* Calc - Current quantiser value quant */
} KlicsEDataRec;

typedef struct {
    KlicsSeqHeader      seqh;
    KlicsFrameHeader    frmh;
    KlicsEDataRec        encd;
    Buffer                buf;
} KlicsERec, *KlicsE;

```

- 657 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

-----
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
-----
*
*  680X0 KlicsDecode code
*  Fast code for:
*    3/2 octave input stream
*    2/1 octave output image
*
-----

```

```

seg      'klics'
include  'Bits3.a'
include  'Traps.a'

```

```

machine  MC68030

```

```

-----
*
*  Data stream readers:
*
*  XDELTA, XVALUE, SKIPHUFF, XINT
*
-----

```

```

macro
XDELTA      &addr,&step,&ptr,&data,&bno,&spare

    buf_rinc    &ptr,&data,&bno      ;
    buf_get     &data,&bno          ;
    beq.s       @quit              ; if zero write
    moveq       #6,&spare           ; set up count
    buf_get     &data,&bno          ; read sign
    bne.s       @doneg             ; if negative -> doneg

@doneg      buf_get     &data,&bno
            dbne       &spare,@dopos ; if --spare!=--1
            bne.s      @findpos

            move.l     &data,&spare  ; spare=data
            subq.b     #7,&bno       ; bno-=6
            lsr.l      &bno,&spare   ; spare>>=bno
            andi.w     #5007F,&spare ; spare AND= mask
            add.w      #8,&spare     ; spare+=9
            bra.s      @write

@findpos    neg.w      &spare        ; bits-=bits
            addq.l     #7,&spare     ; bits+=8
            bra.s      @write

@doneg      buf_get     &data,&bno
            dbne       &spare,@doneg ; if --spare!=--1
            bne.s      @findneg

            move.l     &data,&spare  ; spare=data
            subq.b     #7,&bno       ; bno-=6
            lsr.l      &bno,&spare   ; spare>>=bno
            andi.w     #5007F,&spare ; spare AND= mask

```

- 658 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

        add.w =      #8,&spare      : spare+=9
        neg.w      &spare
        bra.s      @write

@findneg subq.l      #7,&spare      : level-=8

@write lsl.w        &step,&spare    : level<<=step
        swap       &step
        add.w      &step,&spare
        swap       &step
        add.w      &spare,&addr    : *addr=delta
@quit
        .
    endm

macro
XVAL0      &addr,&step,&ptr,&data,&bno,&spare

        clr.w      &spare
        buf_rinc   &ptr,&data,&bno
        buf_get    &data,&bno
        beq.s      @quit          : if zero write
        moveq      #6,&spare      : set up count
        buf_get    &data,&bno      : read sign
        bne.s      @doneg        : if negative -> doneg

@doneg     buf_get   &data,&bno
           dbne      &spare,@doneg : if --spare!=--1
           bne.s     @findpos

           move.l    &data,&spare   : spare=data
           subq.b    #7,&bno        : bno-=6
           lsr.l     &bno,&spare    : spare>>=bno
           andi.w    #5007F,&spare : spare AND= mask
           add.w     #8,&spare      : spare+=9
           bra.s     @write

@findpos   neg.w     &spare          : bits-=bits
           addq.l    #7,&spare      : bits+=8
           bra.s     @write

@doneg     buf_get   &data,&bno
           dbne      &spare,@doneg : if --spare!=--1
           bne.s     @findneg

           move.l    &data,&spare   : spare=data
           subq.b    #7,&bno        : bno-=6
           lsr.l     &bno,&spare    : spare>>=bno
           andi.w    #5007F,&spare : spare AND= mask
           add.w     #8,&spare      : spare+=9
           neg.w     &spare
           bra.s     @write

@findneg   subq.l    #7,&spare      : level-=8

@write     lsl.w     &step,&spare    : level<<=step
           swap      &step
           add.w     &step,&spare
           swap      &step
           move.w     &spare,&addr  : *addr=level
@quit
        .
    endm

```

- 659 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

macro
XVAL1      &addr,&step,&ptr,&data,&bno,&spare

    clr.w      &spare
    buf_rinc   &ptr,&data,&bno
    buf_get    &data,&bno
    beq.s      @quit      ; if zero write
    moveq      #6,&spare   ; set up count
    buf_get    &data,&bno   ; read sign
    bne.s      @doneg     ; if negative -> doneg

@doneg      buf_get    &data,&bno
            dbne      &spare,@doneg      ; if --spare!= -1
            bne.s     @findpos

            move.l    &data,&spare      ; spare=data
            subq.b    #7,&bno          ; bno-=6
            lsr.l     &bno,&spare       ; spare>>=bno
            andi.w    #5007F,&spare    ; spare AND= mask
            add.w     #8,&spare        ; spare+=9
            bra.s     @write

@findpos    neg.w     &spare            ; bits-=bits
            addq.l    #7,&spare        ; bits+=8
            bra.s     @write

@doneg      buf_get    &data,&bno
            dbne      &spare,@doneg    ; if --spare!= -1
            bne.s     @findneg

            move.l    &data,&spare      ; spare=data
            subq.b    #7,&bno          ; bno-=6
            lsr.l     &bno,&spare       ; spare>>=bno
            andi.w    #5007F,&spare    ; spare AND= mask
            add.w     #8,&spare        ; spare+=9
            neg.w     &spare
            bra.s     @write

@findneg    subq.l     #7,&spare        ; level-=8

@write      lsl.w     &step,&spare      ; level<=step
            @quit     move.w    &spare,&addr ; *addr=level

    endm

macro
SKIPHUFF      &ptr,&data,&bno,&spare

    buf_get    &data,&bno
    beq.s      @quit      ; if zero quit
    buf_get    &data,&bno
    moveq      #6,&spare   ; set up count

@do      buf_get    &data,&bno
            dbne      &spare,@do      ; if --spare!= -1
            bne.s     @end

            subq.b    #7,&bno          ; bno-=6
            buf_rinc  &ptr,&data,&bno  ; fill buffer
            @quit

    endm

```

- 660 -

Engineering:KlicsCode:CompPict:KlicsDec2.e

```

macro
XINTX      &bits,&addr,&step,&ptr,&data,&bno
.
: Note: half_q is missing
.
      buf_rinc      &ptr,&data,&bno      ;
      move.l        &data,d0            ; result=data
      sub.b         &bits,&bno          ; dl=bits-1
      subq.b        #1,&bno             ; dl-=1
      lsr.l         &bno,d0             ; result>>=bno
      clr.l         dl                 ; dl=0
      bset          &bits,dl           ; dl[bits]=1
      subq.l        #1,dl              ; dl=mask
      btst          &bits,d0           ; sign?
      beq.s         @pos               ; if positive goto pos
      and.l         dl,d0              ; apply mask leaving level
      neg.l         d0                 ; level=-level
      bra.s         @cont              ; goto cont
@pos      and.l         dl,d0              ; apply mask leaving level
@cont     lsl.l         &step,d0         ; level<<=step
      move.w        d0,&addr           ; *addr=result
.
      endm

macro
XINT      &bits,&addr,&step,&ptr,&data,&bno
.
: Hardware compatible version: sign mag(lsb->msb)
.
      buf_rinc      &ptr,&data,&bno      ;
      move.l        &data,d0            ; result=data
      sub.b         &bits,&bno          ; dl=bits-1
      subq.b        #1,&bno             ; dl-=1
      lsr.l         &bno,d0             ; temp>>=bno
      clr.l         dl                 ; result=0
      swap          &bno                ; use free word
      move.w        &bits,&bno          ; bno=bno.bits
      subq.w        #1,&bno             ; count=bits-2
@shift    lsr.l         #1,d0            ; shift msb from temp
          roxl.l      #1,dl             ; into lsb of result
          dbf         &bno,@shift       ; for entire magnitude
          swap        &bno              ; restore bno
          btst        #0,d0             ; sign test
          beq.s       @pos              ; if positive -> pos
          neg.l       dl                ; result= -result
@pos      lsl.l         &step,dl         ; result<<=step
      move.w        dl,&addr           ; *addr=result
.
      endm

.....
: Block data read/write:
:
: VOID, STILL, SEND, LPPSTILL
:
.....

macro
VOID      &x_blk,&y_blk
.
      clr.w        (a2)

```

- 661 -

Engineering:KlicsCode:CompFict:KlicsDec2.a

```

addq.l    =>  &x_blk,a2          : caddr+=x_blk
clr.w     (&a2)
adda.w    &y_blk,a2             : caddr+=y_blk
clr.w     (&a2)
addq.l    &x_blk,a2             : caddr+=x_blk
clr.w     (&a2)

endm

macro
STILL      &x_blk, &y_blk, &step

XVAL0      (&a2),&step,a0,d6,d7,d0
addq.l     &x_blk,a2             : caddr+=x_blk
XVAL0      (&a2),&step,a0,d6,d7,d0
adda.w     &y_blk,a2             : caddr+=y_blk
XVAL0      (&a2),&step,a0,d6,d7,d0
addq.l     &x_blk,a2             : caddr+=x_blk
XVAL0      (&a2),&step,a0,d6,d7,d0

endm

macro
STILLSEND  &x_blk, &y_blk, &step

XVAL1      (&a2),&step,a0,d6,d7,d0
addq.l     &x_blk,a2             : caddr+=x_blk
XVAL1      (&a2),&step,a0,d6,d7,d0
adda.w     &y_blk,a2             : caddr+=y_blk
XVAL1      (&a2),&step,a0,d6,d7,d0
addq.l     &x_blk,a2             : caddr+=x_blk
XVAL1      (&a2),&step,a0,d6,d7,d0

endm

macro
SEND        &x_blk,&y_blk,&step

XDELTA      (&a2),&step,a0,d6,d7,d0
addq.l     &x_blk,a2             : caddr+=x_blk
XDELTA      (&a2),&step,a0,d6,d7,d0
adda.w     &y_blk,a2             : caddr+=y_blk
XDELTA      (&a2),&step,a0,d6,d7,d0
addq.l     &x_blk,a2             : caddr+=x_blk
XDELTA      (&a2),&step,a0,d6,d7,d0

endm

macro
LPFSTILL    &x_blk, &y_blk, &step, &bits

XINT        &bits, (&a2),&step,a0,d6,d7 : ReadInt (at baddr)
addq.l     &x_blk,a2             : caddr+=x_blk
XINT        &bits, (&a2),&step,a0,d6,d7 : ReadInt
adda.w     &y_blk,a2             : caddr+=y_blk
XINT        &bits, (&a2),&step,a0,d6,d7 : ReadInt
addq.l     &x_blk,a2             : caddr+=x_blk
XINT        &bits, (&a2),&step,a0,d6,d7 : ReadInt

endm

```

.....

- 662 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

*   Data skipping:
*
*   SKIP4, STILLSKIP, SS_SKIP, SENDSKIP
*
*   .....

```

```

SKIP4  FUNC      EXPORT

```

```

      buf_rinc    a0.d6.d7          ; fill buffer
      SKIPHUFF    a0.d6.d7.d0
      SKIPHUFF    a0.d6.d7.d0
      SKIPHUFF    a0.d6.d7.d0
      SKIPHUFF    a0.d6.d7.d0
      rts

```

```

      ENDFUNC

```

```

STILLSKIP  FUNC      EXPORT

```

```

      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @sk1             ; if 0 the STOP
      bsr         SKIP4
      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @sk2             ; if 0 the STOP
      bsr         SKIP4
      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @sk3             ; if 0 the STOP
      bsr         SKIP4
      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @nxt             ; if 0 the STOP
      bsr         SKIP4
      @nxt        rts

```

```

      ENDFUNC

```

```

SS_SKIP  FUNC      EXPORT

```

```

      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @sk1             ; if 0 then STOP
      buf_get     d6.d7             ; BUF_GET
      bne.s       @sk1             ; if 1 then VOID
      bsr         SKIP4
      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @sk2             ; if 0 then STOP
      buf_get     d6.d7             ; BUF_GET
      bne.s       @sk2             ; if 1 then VOID
      bsr         SKIP4
      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @sk3             ; if 0 then STOP
      buf_get     d6.d7             ; BUF_GET
      bne.s       @sk3             ; if 1 then VOID
      bsr         SKIP4
      buf_rinc    a0.d6.d7          ; BUF_INC
      buf_get     d6.d7             ; BUF_GET
      beq.s       @nxt             ; if 0 then STOP
      buf_get     d6.d7             ; BUF_GET

```

- 663 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

    bne.s    @nxt    @nxt    ; if 1 then VOID
    bsr      SKIP4
    rts
@nxt
ENDFUNC

SENDSKIP    FUNC    EXPORT
    buf_rinc    a0,d6,d7    ; BUF_INC
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk1        ; if 0 the STOP
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk0        ; if 0 then STILLSEND
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk1        ; if 0 then VOID

@sk0    bsr      SKIP4
    buf_rinc    a0,d6,d7    ; BUF_INC

@sk1    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk3        ; if 0 the STOP
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk2        ; if 0 then STILLSEND
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk3        ; if 0 then VOID

@sk2    bsr      SKIP4
    buf_rinc    a0,d6,d7    ; BUF_INC

@sk3    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk5        ; if 0 the STOP
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk4        ; if 0 then STILLSEND
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk5        ; if 0 then VOID

@sk4    bsr      SKIP4
    buf_rinc    a0,d6,d7    ; BUF_INC

@sk5    buf_get     d6,d7       ; BUF_GET
    beq.s       @nxt        ; if 0 then STOP
    buf_get     d6,d7       ; BUF_GET
    beq.s       @sk6        ; if 0 then STILLSEND
    buf_get     d6,d7       ; BUF_GET
    beq.s       @nxt        ; if 0 then VOID

@sk6    bsr      SKIP4
@nxt    rts
ENDFUNC

```

```

.....
*
*   Octave Processing:
*
*   DOSTILLO, DOSEND0, DOSTILL1,
*   DOVOID1, DOSTILLSEND1, DOSEND1
*
.....

```

```

DOSTILLO    FUNC    EXPORT

```


- 664 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

buf_rinc    a0,d6,d7      : BUF_INC
buf_get     d6,d7        : BUF_GET
bne.s       @still      : if 1 the STILL
rts

?still move.l    a1,a2      : caddr=baddr

        STILL          #4,d5,d3

XVAL0      (a2),d3,a0,d6,d7,d0
addq.l      #4,a2          : caddr+=x_blk
XVAL0      (a2),d3,a0,d6,d7,d0
adda.w      d5,a2          : caddr+=y_blk
XVAL0      (a2),d3,a0,d6,d7,d0
addq.l      #4,a2          : caddr+=x_blk
XVAL0      (a2),d3,a0,d6,d7,d0

bsr        STILLSKIP
rts

ENDFUNC

DOSEND0 FUNC      EXPORT

        buf_rinc    a0,d6,d7      : BUF_INC
        buf_get     d6,d7        : BUF_GET
        bne.s       @cont        : if 1 then continue
        rts

@cont move.l    a1,a2      : caddr=baddr
        buf_get     d6,d7        : BUF_GET
        beq.w       @ss        : if 0 then STILLSEND
        buf_get     d6,d7        : BUF_GET
        beq.w       @vd        : if 0 then VOID

        SEND        #4,d5,d3

XDELTA      (a2),d3,a0,d6,d7,d0
addq.l      #4,a2          : caddr+=x_blk
XDELTA      (a2),d3,a0,d6,d7,d0
adda.w      d5,a2          : caddr+=y_blk
XDELTA      (a2),d3,a0,d6,d7,d0
addq.l      #4,a2          : caddr+=x_blk
XDELTA      (a2),d3,a0,d6,d7,d0

bsr        SENDSKIP
rts

@ss :STILLSEND #4,d5,d3

XVAL1      (a2),d3,a0,d6,d7,d0
addq.l      #4,a2          : caddr+=x_blk
XVAL1      (a2),d3,a0,d6,d7,d0
adda.w      d5,a2          : caddr+=y_blk
XVAL1      (a2),d3,a0,d6,d7,d0
addq.l      #4,a2          : caddr+=x_blk
XVAL1      (a2),d3,a0,d6,d7,d0

bsr        SS_SKIP
rts

@vd :VOID      #4,d5

```

- 665 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

        clr.w    (a2)
        addq.l   #4,a2                : caddr+=x_blk
        clr.w    (a2)
        adda.w   d5,a2                : caddr+=y_blk
        clr.w    (a2)
        addq.l   #4,a2                : caddr+=x_blk
        clr.w    (a2)
        rts

    ENDFUNC

    macro
    DOSTILL1    &addr

        buf_get    d6,d7                ; BUF_GET
        beq.w      @next                ; if 0 the STOP

        move.l     a1,a2                : caddr=baddr
        add.l       &addr,a2            : caddr+=addrs[1]
        STILL      #4,d5,d4
        bsr        STILLSKIP
        buf_rinc   a0,d6,d7            ; BUF_INC
    @next

    endm

    macro
    DOVOID1     &addr

        move.l     a1,a2                : caddr=baddr
        add.l       &addr,a2            : caddr+=addrs[1]
        VOID       #4,d5

    endm

    macro
    DOSTILLSEND1 &addr

        buf_get    d6,d7                ; BUF_GET
        beq.w      @next                ; if 0 the STOP --
        move.l     a1,a2                : caddr=baddr
        add.l       &addr,a2            : caddr+=addrs[1]
        buf_get    d6,d7                ; BUF_GET
        beq.s      @ss                  ; if 0 then STILLSEND

        VOID       #4,d5
        bra        @next

    @ss    STILLSEND    #4,d5,d4
        bsr        SS_SKIP
        buf_rinc   a0,d6,d7            ; BUF_INC
    @next

    endm

DOSTILL2    FUNC    EXPORT

        buf_rinc   a0,d6,d7            : BUF_INC
        buf_get    d6,d7                : BUF_GET
        bne.s      @cont                : if 1 the CONT
        rts

    @cont    move.l   a1,a2                : caddr=baddr

```

- 666 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

add.l    (a3).a2
STILL    #8.d5.d3      : caddr+=addrs[0]

swap     d5
exg      d4.a5

buf_rinc a0.d6.d7      : BUF_INC
DOSTILL1 4(a3)
DOSTILL1 8(a3)
DOSTILL1 12(a3)
DOSTILL1 16(a3)

swap     d5
exg      d4.a5
rts

macro
DOSEND1  baddr

    buf_get: d6.d7      : BUF_GET
    beq.w    @next      : if 0 the STOP
    move.l   a1.a2      : caddr=baddr
    add.l    baddr,a2   : caddr+=addrs[1]
    buf_get  d6.d7      : BUF_GET
    beq.w    @ss        : if 0 then STILLSEND
    buf_get  d6.d7      : BUF_GET
    beq.w    @vd        : if 0 then VOID

    SEND     #4.d5.d4
    bsr      SENDSKIP
    bra      @rinc

@vd      VOID          #4.d5
bra      @next

@ss      STILLSEND     #4.d5.d4
bsr      SS_SKIP
@rinc    buf_rinc      a0.d6.d7      : BUF_INC
@next

    endm

DOSEND2 FUNC    EXPORT

    buf_rinc     a0.d6.d7      : BUF_INC
    buf_get      d6.d7      : BUF_GET
    bne.s        @cont      : if 1 the CONT
@next    rts

@cont    move.l   a1.a2      : caddr=baddr
          add.l    (a3).a2   : caddr+=addrs[0]
          buf_get  d6.d7      : BUF_GET
          beq.w    @ss      : if 0 then STILLSEND
          buf_get  d6.d7      : BUF_GET
          beq.w    @vd      : if 0 then VOID

*** SEND ***

    SEND         #8.d1.d3

    buf_rinc     a0.d6.d7      : BUF_INC
    DOSEND1      4(a3)
    DOSEND1      8(a3)

```

- 667 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

DOSEND1    = 12(a3)
DOSEND1    = 16(a3)
rts

```

*** STILLSEND ***

3ss STILLSEND =8.d1.d3

```

buf_rinc    a0,d6,d7          : BUF_INC
DOSTILLSEND1 4(a3)
DOSTILLSEND1 8(a3)
DOSTILLSEND1 12(a3)
DOSTILLSEND1 16(a3)
rts

```

*** VOID ***

3vd VOID #8,d1

```

DOVOID1     4(a3)
DOVOID1     8(a3)
DOVOID1     12(a3)
DOVOID1     16(a3)
rts

```

ENDFUNC

```

macro
UVSTILLO

```

* Low_Pass

```

move.l      a1,a2              ; caddr=baddr
LPPSTILL    #4,d5,d2,d4

```

* Sub-band gh

```

addq.l      #2,a1              ; baddr+=2 (gh band)
bsr         DOSTILLO

```

* Sub-band hg

```

subq.l      #2,a1              ; baddr-=2 (hh band)
add.l       a4,a1              ; caddr+=1 row (hg band)
bsr         DOSTILLO

```

* Sub-band gg

```

addq.l      #2,a1              ; baddr+=2 (gg band)
bsr         DOSTILLO
sub.l       a4,a1              ; caddr-=1 row (gh band)
addq.l      #6,a1              ; (2+) addr[0]+=x_inc

```

endm

```

macro
UVSEND0

```

* Low_Pass

```

buf_rinc    a0,d6,d7          : BUF_INC
buf_get     d6,d7             : BUF_GET
beq.w       @subs             ; if 0 then process subbands

```

- 668 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

        move.l  =>  a1,a2                ; caddr=baddr
        SEND      #4,d5,d2
    .
    .   Sub-band gh
    .
2subs    addq.l    #2,a1                ; baddr+=2 (gh band)
        bsr      DOSEND0
    .
    .   Sub-band hg
    .
        subq.l    #2,a1                ; baddr-=2 (hh band)
        add.l     a4,a1                ; caddr+=1 row (hg band)
        bsr      DOSEND0
    .
    .   Sub-band gg
    .
        addq.l    #2,a1                ; baddr+=2 (gg band)
        bsr      DOSEND0
        sub.l     a4,a1                ; caddr-=1 row (gh band)
        addq.l    #6,a1                ; (2+) addr[0]-=x_inc
    .
        endm

.....
    .
    .   Decoder functions:
    .
    .   Klics2D1Still, Klics2D1Send
    .
    .
    .
Klics2D1Still  FUNC    EXPORT
    .
    .   Klics2D1Still(short *dst, long size_x, long size_y, long lpfbits, short *norms
    .
PS        RECORD      8
dst       DS.L         1
size_x    DS.L         1
size_y    DS.L         1
lpfbits   DS.L         1
norms     DS.L         1
ptr       DS.L         1
data      DS.L         1
bno       DS.L         1
        ENDR
    .
LS        RECORD      0,DECR
x_lim     DS.L         1                ; x counter termination      row_start+
x_inc     DS.L         1                ; x termination increment    1 row
y_inc0    DS.L         1                ; y counter increment        4 rows
y_inc1    DS.L         1                ; y counter increment        7 rows
y_lim     DS.L         1                ; y counter termination      area
LSize     EQU          *
        ENDR
    .
    .   d0/d1 - spare
    .   d2 - step 0 (HH)
    .   d3 - step 0
    .   d4 - lpfbits
    .   d5 - y_blk
    .   d6 - data      (bit stream)
    .   d7 - bno       (bit pointer)
    .

```

- 669 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

* a0 - ptr      (bit buffer)
* a1 - baddr    (block address)
* a2 - caddr    (coeff address)
* a3 - x_lim
* a4 - x_linc
* a5 - y_inc0

link          a6,*LS.LSize      ; locals
movem.l       d4-d7/a3-a5,-(a7) ; store registers

*
* Load Bit Buffer
*
move.l        PS.data(a6),a0    ; a0=&data
move.l        (a0),d6           ; data=*a0
move.l        PS.bno(a6),a0     ; a0=&mask
move.l        (a0),d7           ; mask=*a0
move.l        PS.ptr(a6),a0     ; a0=&ptr
move.l        (a0),a0           ; a0=ptr

*
* Set Up Block Counters
*
move.l        PS.dst(a6),a1     ; a1=image
move.l        PS.size_x(a6),d0  ; d0=size_x
add.l         d0,d0             ; in shorts
move.l        d0,LS.x_linc(a6)  ; x_linc=1 row
move.l        PS.size_y(a6),d1  ; d1=size_y
mul.s.w       d0,d1             ; d1=d0 (area)
add.l         a1,d1             ; d1=image
move.l        d1,LS.y_lim(a6)   ; y_lim=d1
move.l        d0,d2             ; d2=d0 (1 row)
add.l         d0,d0             ; d0*=2 (2 rows)
move.l        d0,d5             ; y_blk=d0
subq.l        #4,d5             ; y_blk-=x_blk
add.l         d0,d0             ; d0*=2 (4 rows)
move.l        d0,LS.y_inc0(a6)  ; y_inc0=d0
add.l         d0,d0             ; d0*=2 (8 rows)
sub.l         d2,d0             ; d0-=d2 (7 rows)
move.l        d0,LS.y_incl(a6)  ; y_incl=d0

move.l        PS.norms(a6),a2   ; GetNorm pointer
move.l        (a2),d2           ; read normal
move.l        4(a2),d3          ; read normal
move.l        PS.lpfbits(a6),d4 ; read lpfbits
move.l        LS.x_linc(a6),a4   ; read x_linc
move.l        LS.y_inc0(a6),a5   ; read y_inc0

ey move.l      a4,a3             ; x_lim=x_linc
add.l         a1,a3             ; x_lim=baddr
ex UVSTILLO
UVSTILLO
add.l         a5,a1             ; (2) addr[0]+=y_inc
cmp.l        LS.y_lim(a6),a1    ; (2+) addr[0]-limit?
bge.w        #last             ; if half height
sub.l         #16,a1            ; pointer=blk(0,1)
UVSTILLO
UVSTILLO
elast sub.l     a5,a1            ; (2) addr[0]+=y_inc

cmp.l        a3,a1             ; (2+) addr[0]-limit?
blt.w        ex               ; (4) if less then loopX
add.l         LS.y_incl(a6),a1  ; (2+) addr[0]+=y_inc
cmp.l        LS.y_lim(a6),a1    ; (2+) addr[0]-limit?
blt.w        ey               ; (4) if less then loopY

```

- 670 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

*
* Save Bit Buffer
*
    move.l    PS.data(a6),a2        ; spare=&data
    move.l    d6,(a2)               ; update data
    move.l    PS.bno(a6),a2         ; spare=&bno
    move.l    d7,(a2)               ; update bno
    move.l    PS.ptr(a6),a2         ; spare=&ptr
    move.l    a0,(a2)               ; update ptr
*
    movem.l   (a7)+,d4-d7/a3-a5     ; restore registers
    unlk      a6                    ; remove locals
    rts                          ; return
*
    ENDFUNC
-----
Klics2D1Send    FUNC    EXPORT
*
* Klics2D1Send(short *dst, long size_x, long size_y, short *norms, unsigned long
*
PS      RECORD      8
dst     DS.L         1
size_x  DS.L         1
size_y  DS.L         1
norms   DS.L         1
ptr     DS.L         1
data    DS.L         1
bno     DS.L         1
        ENDR
*
LS      RECORD      0,DECR
x_lim   DS.L         1
x_linc  DS.L         1
y_inc0  DS.L         1
y_incl  DS.L         1
y_lim   DS.L         1
LSize   EQU          *
        ENDR
*
* d0/d1 - spare
* d2 - step 0 (MH)
* d3 - step 0
* d4 - y_inc0
* d5 - y_blk
* d6 - data (bit stream)
* d7 - bno (bit pointer)
*
* a0 - ptr (bit buffer)
* a1 - baddr (block address)
* a2 - caddr (coeff address)
* a3 - x_lim
* a4 - x_linc
* a5 - y_lim
*
    link      a6,#LS.LSize          ; locals
    movem.l   d4-d7/a3-a5,-(a7)     ; store registers
*
* Load Bit Buffer
*
    move.l    PS.data(a6),a0        ; a0=&data
    move.l    (a0),d6               ; data=*a0
    move.l    PS.bno(a6),a0         ; a0=&mask
    move.l    (a0),d7               ; mask=*a0

```

- 671 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

move.l    PS.ptr(a6),a0      ; a0=&ptr
move.l    (a0),a0            ; a0=ptr

: Set Up Block Counters

move.l    PS.dst(a6),a1      ; a1=image
move.l    PS.size_x(a6),d0    ; d0=size_x
add.l     d0,d0              ; in shorts
move.l    d0,LS.x_linc(a6)    ; x_linc=1 row
move.l    PS.size_y(a6),d1    ; d1=size_y
muls.w    d0,d1              ; d1*=d0 (area)
add.l     a1,d1              ; d1+=image
move.l    d1,LS.y_lim(a6)     ; y_lim=d1
move.l    d0,d2              ; d2=d0 (1 row)
add.l     d0,d0              ; d0*=2 (2 rows)
move.l    d0,d5              ; copy to d5
subq.l    #4,d5              ; subtract x_blk
add.l     d0,d0              ; d0*=2 (4 rows)
move.l    d0,LS.y_incl(a6)    ; y_incl=d0
add.l     d0,d0              ; d0*=2 (8 rows)
sub.l     d2,d0              ; d0-=d2 (7 rows)
move.l    d0,LS.y_incl(a6)    ; y_incl=d0

move.l    PS.norms(a6),a2     ; GetNorm pointer
move.l    (a2),d2             ; read normal
move.l    4(a2),d3            ; read normal
move.l    LS.x_linc(a6),a4     ; read x_linc
move.l    LS.y_incl(a6),d4     ; read y_incl
move.l    LS.y_lim(a6),a5     ; read y_lim

ey move.l    a4,a3            ; x_lim=x_linc
add.l     a1,a3              ; x_lim+=baddr
ex UVSEND0    ; process UV block 0,0
UVSEND0    ; process UV block 1,0
add.l     d4,a1              ; (2) addr[0]+=y_incl
cmp.l     a5,a1              ; (2) addr[0]-limit?
bge.w     @last              ; if half height
sub.l     #16,a1             ; pointer=blk(0,1)
UVSEND0    ; process UV block 0,1
UVSEND0    ; process UV block 1,1
@last sub.l     d4,a1         ; (2) addr[0]+=y_incl

cmp.l     a3,a1              ; (2) addr[0]-limit?
blt.w     ex                 ; (4) if less then loopX
add.l     LS.y_incl(a6),a1    ; (2+) addr[0]+=y_incl
cmp.l     a5,a1              ; (2) addr[0]-limit?
blt.w     ey                 ; (4) if less then loopY

: Save Bit Buffer

move.l    PS.data(a6),a2      ; spare=&data
move.l    d6,(a2)             ; update data
move.l    PS.bno(a6),a2       ; spare=&bno
move.l    d7,(a2)             ; update bno
move.l    PS.ptr(a6),a2       ; spare=&ptr
move.l    a0,(a2)             ; update ptr

movem.l    (a7)+,d4-d7/a3-a5   ; restore registers
unlk      a6                  ; remove locals
rts                          ; return

ENDFUNC
-----

```


- 672 -

Engineering:KlacsCode:CompPict:KlacsDec2.a

Klacs3D2Still1 FUNC EXPORT

: Klacs3D2Still1(short *dst, long size_x, long size_y, long lpfbits, short *norms

```

PS      RECORD      8
dst      DS.L        1
size_x   DS.L        1
size_y   DS.L        1
lpfbits  DS.L        1
norms    DS.L        1
ptr      DS.L        1
data     DS.L        1
bnc      DS.L        1
sub_tab  DS.L        1
        ENDR

```

```

LS      RECORD      0,DECR
y_blk0   DS.L        1          ; y inter-block increment 2 rows - 4
y_blk1   DS.L        1          ; y inter-block increment 4 rows - 8
x_inc    DS.L        1          ; x counter increment 16
x_lim    DS.L        1          ; x counter termination row_start+
x_linc   DS.L        1          ; x termination increment 1 row
y_inc    DS.L        1          ; y counter increment 7 rows
y_lim    DS.L        1          ; y counter termination area
LSize    EQU         *
        ENDR

```

```

: d0/d1 - spare
: d2 - step 2HH
: d3 - step 1
: d4 - step 0/lpfbits
: d5 - y_blk0,y_blk1
: d6 - data (bit stream)
: d7 - bnc (bit pointer)
:
: a0 - ptr (bit buffer)
: a1 - baddr (block address)
: a2 - caddr (coeff address)
: a3 - addr (tree addresses)
: a4 - x_lim (x counter termination)
: a5 - lpfbits/step 0

```

```

link      a6, #LS.LSize      ; locals
movem.l   d4-d7/a3-a5, -(a7) ; store registers

```

: Load Bit Buffer

```

move.l    PS.data(a6),a0      ; a0=&data
move.l    (a0),d6             ; data=*a0
move.l    PS.bnc(a6),a0       ; a0=&mask
move.l    (a0),d7             ; mask=*a0
move.l    PS.ptr(a6),a0       ; a0=&ptr
move.l    (a0),a0             ; a0=ptr

```

: Set Up Block Counters

```

move.l    PS.dst(a6),a1       ; a1=image
move.l    PS.size_x(a6),d0     ; d0=size_x
move.l    #16,LS.x_inc(a6)    ; save x_inc
add.l     d0,d0               ; in shorts
move.l    d0,LS.x_linc(a6)    ; x_linc=1 row
move.l    PS.size_y(a6),d1     ; d1=size_y
muls.w    d0,d1               ; d1*=d0 (area)

```

- 673 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

add.l    a1,d1                ; d1:=image
move.l   d1,LS.y_lim(a6)      ; y_lim=d1
move.l   d0,d2                ; d2=d0 (1 row)
add.l    d0,d0                ; d0*=2 (2 rows)
move.l   d0,d5                ; copy to d5
subq.l   #4,d5                ; y_blk: subtract x_blk
move.l   d5,LS.y_blk0(a6)     ; save y_blk0
add.l    d0,d2                ; d2+=d0 (3 rows)
add.l    d0,d0                ; d0*=2 (4 rows)
move.l   d0,d4                ; copy to d5
subq.l   #8,d4                ; y_blk: subtract x_blk
move.l   d4,LS.y_blk1(a6)     ; save y_blk1
add.l    d2,d0                ; d0+=d2 (7 rows)
move.l   d0,LS.y_inc(a6)      ; y_inc=d0

move.l   PS.norms(a6),a2      ; GetNorm pointer
move.l   (a2),d2              ; read normal
move.l   4(a2),d3             ; read normal 1
move.l   8(a2),a5             ; read normal 0
move.l   PS.lpfbits(a6),d4    ; read lpfbits
swap     d5                  ; y_blk=00XX
move.l   LS.y_blk1(a6),d0      ; read y_blk1
move.w   d0,d5                ; d5=y_blk0/1
move.l   PS.sub_tab(a6),a3     ; a3=addr

@y      move.l   LS.x_linc(a6),a4 ; x_lim=x_linc
add.l    a1,a4                ; x_lim+=baddr
.
.      Low_Pass
.
@x      move.l   a1,a2          ; caddr=baddr
LPPSTILL #8,d5,d2,d4
.
.      Sub-band gh
.
      bsr       DOSTILL2
add.l    #20,a3
.
.      Sub-band hg
.
      bsr       DOSTILL2
add.l    #20,a3
.
.      Sub-band gg
.
      bsr       DOSTILL2
sub.l    #40,a3

add.l    #16,a1                ; (2) addr[0]+=x_inc
cmp.l    a4,a1                ; (2) addr[0]-limit?
blt.w    @x                   ; (4) if less then loopX
add.l    LS.y_inc(a6),a1       ; (2+) addr[0]+=y_inc
cmp.l    LS.y_lim(a6),a1       ; (2+) addr[0]-limit?
blt.w    @y                   ; (4) if less then loopY
.
.      Save Bit Buffer
.
@end     move.l   PS.data(a6),a2 ; spare=&data
move.l   d6,(a2)              ; update data
move.l   PS.bno(a6),a2        ; spare=&bno
move.l   d7,(a2)              ; update bno
move.l   PS.ptr(a6),a2        ; spare=&ptr
move.l   a0,(a2)              ; update ptr

```

- 674 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

Page 18

```

    movem.l    (a7)-,d4-d7/a3-a5    ; restore registers
    unlk      a6                    ; remove locals
    rts                          ; return

    ENDFUNC
-----
Klics3D2Send    FUNC    EXPORT
    Klics3D2Send(short *dst, long size_x, long size_y, short *norms, unsigned long
PS      RECORD      8
dst      DS.L        1
size_x   DS.L        1
size_y   DS.L        1
norms    DS.L        1
ptr      DS.L        1
data     DS.L        1
bno      DS.L        1
sub_tab  DS.L        1
    ENDR

LS      RECORD      0,DECR
y_blk0   DS.L        1            ; y inter-block increment    2 rows - 4
y_blk1   DS.L        1            ; y inter-block increment    4 rows - 8
x_inc    DS.L        1            ; x counter increment        16
x_lim    DS.L        1            ; x counter termination      row_start+
x_linc   DS.L        1            ; x termination increment    1 row
y_inc    DS.L        1            ; y counter increment        7 rows
y_lim    DS.L        1            ; y counter termination      area
LSize    EQU         *
    ENDR

    d0 - spare
    d1 - y_blk1
    d2 - step 2HH
    d3 - step 1
    d4 - step 0
    d5 - y_blk0
    d6 - data      (bit stream)
    d7 - bno       (bit pointer)

    a0 - ptr       (bit buffer)
    a1 - baddr     (block address)
    a2 - caddr     (coeff address)
    a3 - addrs     (tree addresses)
    a4 - x_lim     (x counter termination)

    link      a6,*LS.LSize          ; locals
    movem.l   d4-d7/a3-a5,-(a7)     ; store registers

    Load Bit Buffer

    move.l    PS.data(a6),a0        ; a0=&data
    move.l    (a0),d6               ; data+*a0
    move.l    PS.bno(a6),a0         ; a0=&mask
    move.l    (a0),d7               ; mask+*a0
    move.l    PS.ptr(a6),a0         ; a0=&ptr
    move.l    (a0),a0               ; a0=ptr

    Set Up Block Counters

    move.l    PS.dst(a6),a1         ; a1=image

```

- 675 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```

move.l    PS.size_x(a6),d0      ; d0=size_x
move.l    #16,LS.x_inc(a6)      ; save x_inc
add.l     d0,d0                 ; in shorts
move.l    d0,LS.x_linc(a6)      ; x_linc=1 row
move.l    PS.size_y(a6),d1      ; d1=size_y
mul.s.w   d0,d1                 ; d1*=d0 (area)
add.l     a1,d1                 ; d1+=image
move.l    d1,LS.y_lim(a6)       ; y_lim=d1
move.l    d0,d2                 ; d2=d0 (1 row)
add.l     d0,d0                 ; d0*=2 (2 rows)
move.l    d0,d5                 ; copy to d5
subq.l    #4,d5                 ; y_blk: subtract x_blk
move.l    d5,LS.y_blk0(a6)      ; save y_blk0
add.l     d0,d2                 ; d2+=d0 (3 rows)
add.l     d0,d0                 ; d0*=2 (4 rows)
move.l    d0,d4                 ; copy to d5
subq.l    #8,d4                 ; y_blk: subtract x_blk
move.l    d4,LS.y_blk1(a6)      ; save y_blk1
add.l     d2,d0                 ; d0+=d2 (7 rows)
move.l    d0,LS.y_inc(a6)       ; y_inc=d0

move.l    PS.norms(a6),a2       ; GetNorm pointer
move.l    (a2),d2               ; read normal
move.l    4(a2),d3              ; read normal 1
move.l    8(a2),d4              ; read normal 0
move.l    LS.y_blk1(a6),d1      ; read y_blk1
move.l    PS.sub_tab(a6),a3     ; a3=addr

@y      move.l    LS.x_linc(a6),a4 ; x_lim=x_linc
add.l     a1,a4                 ; x_lim+=baddr

:      Low_Pass

ex      buf_rinc    a0,d6,d7      ; BUF_INC
buf_get  d6,d7                  ; BUF_GET
beq.w    #subs      ; if 0 then process subbands
move.l    a1,a2
SEND     #8,d1,d2
:
:      Sub-band gh
:
@subs    bsr        DOSEND2
add.l     #20,a3
:
:      Sub-band hg
:
        bsr        DOSEND2
add.l     #20,a3
:
:      Sub-band gg
:
        bsr        DOSEND2
sub.l     #40,a3

add.l     #16,a1                ; (2) addr[0]+=x_inc
cmp.l     a4,a1                 ; (2) addr[0]-limit?
blt.w     0x                    ; (4) if less then loopX
add.l     LS.y_inc(a6),a1       ; (2+) addr[0]+=y_inc
cmp.l     LS.y_lim(a6),a1       ; (2+) addr[0]-limit?
blt.w     0y                    ; (4) if less then loopY

:      Save Bit Buffer
:

```

- 676 -

Engineering:KlicsCode:CompPict:KlicsDec2.a

```
3end      move.l    PS.data(a6),a2      : spare=&data
          move.l    d6,(a2)              : update data
          move.l    PS.bno(a6),a2       : spare=&bno
          move.l    d7,(a2)              : update bno
          move.l    PS.ptr(a6),a2        : spare=&ptr
          move.l    a0,(a2)              : update ptr
          .
          movem.l   (a7)+,d4-d7/a3-a5    : restore registers
          unlink    a6                   : remove locals
          rts                               : return
          .
          ENDFUNC
          -----
          END
```

Engineering:KlicsCode:CompPict:KlicsDec.c

```

/*****
 *
 * Copyright 1993 KLICS Limited
 * All rights reserved.
 *
 * Written by: Adrian Lewis
 *
 *****/

/*
 * Importing raw Klics binary files
 *
 * Stand-alone version
 */

#include "Bits3.h"
#include "Klics.h"
#include "KlicsHeader.h"

typedef char Boolean;

/* If bool true the negate value */
#define negif(bool,value) ((bool)?-(value):(value))

extern void HaarBackward();
extern void Daub4Backward(short *data,int size[2],int oct_src);
extern void TestTopBackward(short *data,int size[2],int oct_src);
extern void TestBackward(short *data,int size[2],int oct_src);
extern void KLICSDCHANNEL(short *dst, long octs, long size_x, long size_y, long
/* Use the bit level file macros (Bits2.h) */
/* buf_use: */

/* Huffman decode a block */
#define HuffDecLev(lev,buf) \
    lev[0]=HuffDecode(buf); \
    lev[1]=HuffDecode(buf); \
    lev[2]=HuffDecode(buf); \
    lev[3]=HuffDecode(buf);

/* Fixed length decode block of integers */
#define IntDecLev(lev,lpf_bits,buf) \
    lev[0]=IntDecode(lpf_bits,buf); \
    lev[1]=IntDecode(lpf_bits,buf); \
    lev[2]=IntDecode(lpf_bits,buf); \
    lev[3]=IntDecode(lpf_bits,buf);

/* Reverse quantize difference block */
#define RevQntDelta(new,old,lev,shift) \
    new[0]=old[0]+(lev[0]<<shift)+(lev[0]!=0?negif(lev[0]<0,(1<<shift)-1)>>1):0); \
    new[1]=old[1]+(lev[1]<<shift)+(lev[1]!=0?negif(lev[1]<0,(1<<shift)-1)>>1):0); \
    new[2]=old[2]+(lev[2]<<shift)+(lev[2]!=0?negif(lev[2]<0,(1<<shift)-1)>>1):0); \
    new[3]=old[3]+(lev[3]<<shift)+(lev[3]!=0?negif(lev[3]<0,(1<<shift)-1)>>1):0);

/* Reverse quantize block */
#define RevQnt(new,lev,shift) \
    new[0]=(lev[0]<<shift)+(lev[0]!=0?negif(lev[0]<0,(1<<shift)-1)>>1):0); \
    new[1]=(lev[1]<<shift)+(lev[1]!=0?negif(lev[1]<0,(1<<shift)-1)>>1):0); \
    new[2]=(lev[2]<<shift)+(lev[2]!=0?negif(lev[2]<0,(1<<shift)-1)>>1):0); \
    new[3]=(lev[3]<<shift)+(lev[3]!=0?negif(lev[3]<0,(1<<shift)-1)>>1):0);

#define RevQntLPP(new,lev,shift) \
    new[0]=(lev[0]<<shift)+((1<<shift)-1)>>1); \
    new[1]=(lev[1]<<shift)+((1<<shift)-1)>>1); \
    new[2]=(lev[2]<<shift)+((1<<shift)-1)>>1); \

```

- 678 -

Engineering:KlicsCode:CompPict:KlicsDec.c

```

new[3]=Flev[3]<<shift)+((1<<shift.-1)>>1);

/* Read a difference block and update memory */
#define DoXferDelta(addr,old,new,lev,dst,shift,mode,oct,nmode,buf) \
    HuffDecLev(lev,buf); \
    RevCntDelta(new,old,lev,shift) \
    PutData(addr,new,dst); \
    mode[oct]=oct==0?M_STOP:nmode;

/* Read a block and update memory */
#define DoXfer(addr,new,lev,dst,shift,mode,oct,nmode,buf) \
    HuffDecLev(lev,buf); \
    RevCnt(new,lev,shift) \
    PutData(addr,new,dst); \
    mode[oct]=oct==0?M_STOP:nmode;

/* Function Name:  IntDecode
 * Description:    Read a integer from bit file
 * Arguments:     bits - bits/integer now signed
 * Returns:       integer value
 */

short  IntDecode(short bits,Buf buf)
{
    int      i, lev=0, mask=1;
    Boolean sign;

    /* Hardware compatible version */
    buf_rinc(buf);
    sign=buf_get(buf);
    for(i=0;i<bits-1;i++) {
        buf_rinc(buf);
        if (buf_get(buf)) lev |= mask;
        mask <<= 1;
    }
    if (sign) lev= -lev;
    return(lev);
}

/* Function Name:  HuffDecode
 * Description:    Read a Huffman coded integer from bit file
 * Returns:       integer value
 */

short  HuffDecode(Buf buf)
{
    short  lev=0, i;
    Boolean neg;

    /* Hardware compatible version */
    buf_rinc(buf);
    if (buf_get(buf)) {
        buf_rinc(buf);
        neg=buf_get(buf);
        do {
            buf_rinc(buf);
            lev++;
        } while (lev<7 && !(buf_get(buf)));
        if (!(buf_get(buf))) {
            for(lev=0,i=0;i<7;i++) {
                lev<<=1;
                buf_rinc(buf);
            }
        }
    }
    if (neg) lev= -lev;
    return(lev);
}

```

- 679 -

Engineering:KlicsCode:CompPict:KlicsDec.c

```

    > if (buf_get(buf)) lev++;
    }
    lev+=8;
}
if (neg) lev= -lev;
}
return(lev);
}

/* Function Name: KlicsDChannel
 * Description: Decode a channel of image
 * Arguments: dst - destination memory (and old for videos)
 *            octs, size - octaves of decomposition and image dimensions
 *            normals - HVS weighted normals
 *            lpf_bits - no of bits for LPF integer (image coding only)
 */

void KlicsDecY(short *dst, int octs, int size[2], KlicsFrameHeader *frmh,
KlicsSeqHeader *seqh, Buf buf)
{
    int oct, mask, x, y, sub, step=2<<octs, blk[4], mode[4], base_mode=(frmh->
Blk addr, new, old, lev;

    for(y=0;y<size[1];y+=step)
    for(x=0;x<size[0];x+=step)
    for(sub=0;sub<4;sub++) {
        mode[oct=octs-1]=base_mode;
        if (sub==0) mode[oct=octs-1] != M_LPF;
        mask=2<<oct;
        do {
            GetAddr(addr,x,y,sub,oct,size,mask);
            switch(mode[oct]) {
                case M_VOID:
                    GetData(addr,old,dst);
                    if (BlkZero(old)) mode[oct]=M_STOP;
                    else { DoZero(addr,dst,mode,oct); }
                    break;
                case M_SEND|M_STILL:
                    buf_rinc(buf);
                    if (buf_get(buf)) {
                        buf_rinc(buf);
                        if (buf_get(buf)) {
                            DoZero(addr,dst,mode,oct);
                        } else {
                            DoXfer(addr,new,lev,dst,frmh->quantizer[octs-oct],mode,oct,M_S
                        )
                    }
                    ) else
                        mode[oct]=M_STOP;
                    break;
                case M_SEND:
                    buf_rinc(buf);
                    if (buf_get(buf)) {
                        buf_rinc(buf);
                        if (buf_get(buf)) {
                            buf_rinc(buf);
                            if (buf_get(buf)) {
                                GetData(addr,old,dst);
                                DoXferDelta(addr,old,new,lev,dst,frmh->quantizer[octs-oct]
                            )
                            ) else {
                                DoZero(addr,dst,mode,oct);
                            }
                        }
                    }
                    ) else {
                        DoXfer(addr,new,lev,dst,frmh->quantizer[octs-oct],mode,oct,M_S
    }
}

```


- 680 -

Engineering:KlicsCode:CompPict:KlicsDec.c

```

    )
    } else
        mode{oct}=M_STOP;
        break;
    case M_STILL:
        buf_rinc(buf);
        if (buf_get(buf)) { DoXfer(addr,new,lev,dst,frmh->quantizer{octs-oct});
        else mode{oct}=M_STOP;
        break;
    case M_LPFIM_STILL:
        IntDecLev(lev,seqh->precision-frmh->quantizer[0],buf);
        RevOntLPF(new,lev,frmh->quantizer[0]);
        PutData(addr,new,dst);
        mode{oct}=M_QUIT;
        break;
    case M_LPFIM_SEND:
        buf_rinc(buf);
        if (buf_get(buf)) {
            GetData(addr,old,dst);
            HuffDecLev(lev,buf);
            RevOntDelta(new,old,lev,frmh->quantizer[0]);
            PutData(addr,new,dst);
        }
        mode{oct}=M_QUIT;
        break;
    }
    switch(mode{oct}) {
    case M_STOP:
        StopCounters(mode,oct,mask,blk,x,y,octs);
        break;
    case M_QUIT:
        break;
    default:
        DownCounters(mode,oct,mask,blk);
        break;
    }
    } while (mode{oct}!=M_QUIT);
}

void KlicsDecUV(short *dst, int octs, int size[2], KlicsFrameHeader *frmh,
KlicsSeqHeader *seqh, Buf buf)
{
    int oct, mask, x, y, X, Y, sub, step=4<<octs, blk[4], mode[4], base_mode;
    Blk addr, new, old, lev;

    for(Y=0;Y<size[1];Y+=step)
    for(X=0;X<size[0];X+=step)
    for(y=Y;y<size[1] && y<Y+step;y+=step>>1)
    for(x=X;x<size[0] && x<X+step;x+=step>>1)
    for(sub=0;sub<4;sub++) {
        mode{oct=octs-1}=base_mode;
        if (sub==0) mode{oct=octs-1} != M_LPF;
        mask=2<<oct;
        do {
            GetAddr(addr,x,y,sub,oct,size,mask);
            switch(mode{oct}) {
            case M_VOID:
                GetData(addr,old,dst);
                if (BlkZero(old)) mode{oct}=M_STOP;
                else { DoZero(addr,dst,mode,oct); }
                break;
            case M_SENDIM_STILL:

```

- 681 -

Engineering:KlicsCode:CompPict:KlicsDec.c

```

    buf_rinc(buf);
    if (buf_get(buf)) {
        buf_rinc(buf);
        if (buf_get(buf)) {
            DoZero(addr,dst,mode,oct);
        } else {
            DoXfer(addr,new.lev,dst,frmh->quantizer(octs-oct),mode,oct,M_S
        )
    } else
        mode[oct]=M_STOP;
    break;
case M_SEND:
    buf_rinc(buf);
    if (buf_get(buf)) {
        buf_rinc(buf);
        if (buf_get(buf)) {
            buf_rinc(buf);
            if (buf_get(buf)) {
                GetData(addr,old,dst);
                DoXferDelta(addr,old,new.lev,dst,frmh->quantizer(octs-oct)
            ) else {
                DoZero(addr,dst,mode,oct);
            }
        } else {
            DoXfer(addr,new.lev,dst,frmh->quantizer(octs-oct),mode,oct,M_S
        )
    } else
        mode[oct]=M_STOP;
    break;
case M_STILL:
    buf_rinc(buf);
    if (buf_get(buf)) { DoXfer(addr,new.lev,dst,frmh->quantizer(octs-oct);
    else mode[oct]=M_STOP;
    break;
case M_LPFIM_STILL:
    IntDecLev(lev,seqh->precision-frmh->quantizer[0],buf);
    RevOntLFF(new,lev,frmh->quantizer[0]);
    PutData(addr,new,dst);
    mode[oct]=M_QUIT;
    break;
case M_LPFIM_SEND:
    buf_rinc(buf);
    if (buf_get(buf)) {
        GetData(addr,old,dst);
        HuffDecLev(lev,buf);
        RevOntDelta(new,old,lev,frmh->quantizer[0]);
        PutData(addr,new,dst);
    }
    mode[oct]=M_QUIT;
    break;
}
switch(mode[oct]) {
case M_STOP:
    StopCounters(mode,oct,mask,blk,x,y,octs);
    break;
case M_QUIT:
    break;
default:
    DownCounters(mode,oct,mask,blk);
    break;
}
} while (mode[oct]!=M_QUIT);
}

```

- 682 -

Engineering:KlicsCode:CompPict::KlicsDec.c

```

)

/* Function Name: KlicsDecode
 * Description:   Decode a frame to YUV (de)transformed image
 * Arguments:    src - destination result
 *              dst - transformed destination memory (and old for videos)
 * Returns:      whether this frame was skipped
 */

extern void KLCOPY(short *dst, short *src, long area);
extern void KLHALF(short *dst, short *src, long size_0, long size_1);
extern void KLICS3D2SEND(short *dst, long size_x, long size_y, short norms[4]);
extern void KLICS2D1STILL(short *dst, long size_x, long size_y, long lpfbits);
extern void KLICS3D2STILL(short *dst, long size_x, long size_y, long lpfbits);
extern void KLICS2D1SEND(short *dst, long size_x, long size_y, short norms[4]);

#define flag_tree 0x1
#define flag_wave 0x2

void KlicsDecode(short *src[3], short *dst[3], KlicsSeqHeader *seqh, KlicsFrameH
(
    long channel, i;
    short norms[4][2];
    unsigned long sync1, sync2;

    for(i=0; i<4; i++) {
        norms[i][0] = (1<<(frmh->quantizer[i]-1))-1;
        norms[i][1] = frmh->quantizer[i];
    }
    buf_rinit(buf);
    if (0!=(flags&flag_tree)) {
        sync1=GetTimerValue(&sync1);
        for(channel=0; channel<seqh->channels; channel++) {
            int size[2] = (seqh->sequence_size[0]>>(channel==0?0:seqh->sub_sampl
                seqh->sequence_size[1]>>(channel==0?0:seqh->sub_sample[1]))
            tree_size[2] = (size[0]>>scale[0], size[1]>>scale[0]),
            octs = seqh->octaves[channel]==0?0:1;

#ifdef HQ
            if (0!=(frmh->flags&KFM_INTRA)) {
                KLZERO(dst[channel], tree_size[0]*tree_size[1]);
                KLICS3DCHANNEL(dst[channel], octs-1, tree_size[0], tree_size[1], (long) (seq
            if (channel==0) KlicsDecY(dst[channel], octs, tree_size, frmh, seqh, buf);
            else KlicsDecUV(dst[channel], octs, tree_size, frmh, seqh, buf);
            }
        else
            long sub_tab[15] = {4, 2, 10, 2+8*tree_size[0], 10+8*tree_size[0],
                4*tree_size[0], 2*tree_size[0], 8+2*tree_size[0], 10*tree_siz
                4+4*tree_size[0], 2+2*tree_size[0], 10+2*tree_size[0], 2+10*t

            if (0!=(frmh->flags&KFM_INTRA)) {
                KLZERO(dst[channel], tree_size[0]*tree_size[1]);
                if (octs==3)
                    KLICS3D2STILL(dst[channel], tree_size[0], tree_size[1], (long) (se
                else
                    KLICS2D1STILL(dst[channel], tree_size[0], tree_size[1], (long) (se
            ) else
                if (octs==3)
                    KLICS3D2SEND(dst[channel], tree_size[0], tree_size[1], &norms, &bu
                else
                    KLICS2D1SEND(dst[channel], tree_size[0], tree_size[1], &norms, &bu
            }
        }
    }
    sync2=GetTimerValue(&sync2);

```

- 683 -

Engineering:KlicsCode:CompPict:KlicsDec.c

```

*tree=sync2-sync1;
)
if (0!==(flags&flag_wave)) {
    sync1=GetTimerValue(&sync1);
    for(channel=0;channel<seqh->channels;channel++) {
        int    size[2]=(seqh->sequence_size[0]>>(channel==0?0:seqh->sub_sampl
            seqh->sequence_size[1]>>(channel==0?0:seqh->sub_sample[1]))
        wave_size[2]=(size[0]>>scale[1],size[1]>>scale[1]),
        octs=seqh->octaves(channel==0?0:1);

        switch(seqh->wavelet) {
        case WT_Haar:
            if (scale[1]>scale[0])
                KLHALF(dst[channel],src[channel],wave_size[0],wave_size[1]);
            else
                KLCOPY(dst[channel],src[channel],wave_size[0]*wave_size[1]);
            HaarBackward(src[channel],wave_size,octs-scale[1]);
            break;
        case WT_Daub4:
            if (scale[0]==0) {
                if (scale[1]>scale[0])
                    KLHALF(dst[channel],src[channel],wave_size[0],wave_size[1])
                else
                    KLCOPY(dst[channel],src[channel],wave_size[0]*wave_size[1])
                Daub4Backward(src[channel],wave_size,octs-scale[1]);
            } else
                if (channel==0) {
                    KLCOPY(dst[channel],src[channel],wave_size[0]*wave_size[1])
                    Backward3511(src[channel],wave_size,octs-scale[1]);
                } else
                    TOPBWD(dst[channel],src[channel],wave_size[0],wave_size[1])
            break;
        }
    }
    sync2=GetTimerValue(&sync2);
    *wave=sync2-sync1;
}

```

- 684 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

.....
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
*...../
/*
*  Klics Codec
*/

#include "ImageCodec.h"
#include <FixMath.h>
#include <Errors.h>
#include <Packages.h>

#ifdef PERFORMANCE
    #include <Perf.h>
    extern IP2PerfGlobals ThePGlobals;
#endif

#ifdef DEBUG
    #define DebugMsg(val)    DebugStr(val)
#else
    #define DebugMsg(val)
#endif

#define WT_Haar  0
#define WT_Daub4 1

#define None      0
#define Use8      1
#define Use16     2
#define Use32     3
#define UseF32    4

/* Version information */

#define KLICS_CODEC_REV      1
#define codecInterfaceVersion 1 /* high word returned in component GetVersion */

#define klicsCodecFormatName  "Klics"
#define klicsCodecFormatType  "klic"

pascal ComponentResult
KlicsCodec(ComponentParameters *params, char **storage);

pascal ComponentResult
KLOpenCodec(ComponentInstance self);

pascal ComponentResult
KLCloseCodec(Handle storage, ComponentInstance self);

pascal ComponentResult
KLCanDoSelector(short selector);

pascal ComponentResult
KLGetVersion();

pascal ComponentResult
KLGetCodecInfo(Handle storage, CodecInfo *info);

```

- 685 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

pascal ComponentResult
KLGetMaxCompressionSize(Handle storage, PixmapHandle src, const Rect *srcRect, short
    CodecQ quality, long *size);

pascal ComponentResult
KLGetCompressedImageSize(Handle storage, ImageDescriptionHandle desc, Ptr data, long
    DataProcRecordPtr dataProc, long *size);

pascal ComponentResult
KLPreCompress(Handle storage, register CodecCompressParams *p);

pascal long
KLPreDecompress(Handle storage, register CodecDecompressParams *p);

pascal long
KLBandDecompress(Handle storage, register CodecDecompressParams *p);

pascal long
KLBandCompress(Handle storage, register CodecCompressParams *p);

pascal ComponentResult
KLGetCompressionTime(Handle storage, PixmapHandle src, const Rect *srcRect, short dep
    CodecQ *spatialQuality, CodecQ *temporalQuality, unsigned long *time);

/* Function: KlicsCodec
 * Description: KlicsCodec main dispatcher
 */

#ifdef DECODER
pascal ComponentResult
KlicsDecoder(ComponentParameters *params, char **storage)
#else
#ifdef ENCODER
pascal ComponentResult
KlicsEncoder(ComponentParameters *params, char **storage)
#else
pascal ComponentResult
KlicsCodec(ComponentParameters *params, char **storage)
#endif
#endif
{
    OSErr err;

    switch ( params->what ) {
        case kComponentOpenSelect:
            err=CallComponentFunction(params, (ComponentFunction) KLOpenCodec); break;

        case kComponentCloseSelect:
            err=CallComponentFunctionWithStorage(storage, params, (ComponentFunction) KLC
            case kComponentCanDoSelect:
                err=CallComponentFunction(params, (ComponentFunction) KLCanDoSelector); brea
            case kComponentVersionSelect :
                err=CallComponentFunction(params, (ComponentFunction) KLGetVersion); break;

#ifdef DECODER
        case codecPreCompress:
        case codecBandCompress:
            err=codecUnimpErr; break;
        else
            case codecPreCompress:

```

- 686 -

Engineering:KlicsCode:CompFict:KlicsCodec.c

```

err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLP

case codecBandCompress:
    err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLB
#endif
#ifdef ENCODER
    case codecPreDecompress:
    case codecBandDecompress:
        err=codecUnimpErr; break;
#else
    case codecPreDecompress:
        err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLP

    case codecBandDecompress:
        err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLB
#endif
case codecCDSequenceBusy:
    err=0; break; /* our codec is never asynchronously busy

case codecGetCodecInfo:
    err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG

case codecGetCompressedImageSize:
    err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG

case codecGetMaxCompressionSize:
    err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG

case codecGetCompressionTime:
    err=CallComponentFunctionWithStorage(storage,params,(ComponentFunction)KLG

case codecGetSimilarity:
    err=codecUnimpErr; break;

case codecTrimImage:
    err=codecUnimpErr; break;

default:
    err=paramErr; break;
}
if (err!=noErr)
    DebugMsg("\pCodec Error");
return(err);
}

#include <Memory.h>
#include <Resources.h>
#include <OSUtils.h>
#include <SysEqu.h>

#include <StdIO.h>
#include <Time.h>

#include <Strings.h>
#include <String.h>
#include "Bits3.h"
#include "KlicsHeader.h"
#include "KlicsEncode.h"

void DebugString(char *string)
{
    DebugStr(string);
}

```

- 687 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

extern short gResRef;

```
typedef struct (
    CodecInfo **info;
    Ptr tab(4);
    short use(4);
) SharedGlobals;
```

```
typedef struct (
    KlicsERec kle;
    short *src[3];
    short *dst[3];
    Ptr pixmap;
    long size;
    long using;
    long scale[3];
    unsigned long prev_frame;
    unsigned long real_frame;
    unsigned long dpy_frame;
    unsigned long run_frame;
    unsigned long sys_time;
    unsigned long tree_time;
    unsigned long wave_time;
    unsigned long dpy_time;
    unsigned long run_time;
    unsigned long key_time;
    unsigned long sync_time;
    Boolean out[15];
    SharedGlobals *sharedGlob;
) Globals;
```

/* Encoding parameters */
 /* YUV Frame buffer */
 /* YUV Frame buffer */
 /* Encoded pixmap data */
 /* Size of Previous Frame Buffer */
 /* Which lookup table are we using for colour
 /* Tree, Wave, Out scales 0=Original, -1=Double
 /* Previous frame number */
 /* Previous real frame (no skips) */
 /* Previous displayed frame */
 /* First frame in play sequence */
 /* System overhead for previous frame */
 /* Typical tree decode time (not skip) */
 /* Typical wavelet transform time */
 /* Typical display time */
 /* Time of first run frame */
 /* Time at last key frame */
 /* Sync time */
 /* Displayed? */

/* Scaling scenarios: Tree Wave Out

```

* 1 1 0: Internal calculations are Quarter size, output Original size (interpo
* 1 1 1: Internal calculations are Quarter size, output Quarter size
* 0 1 1: Internal calculations are Original size, output Quarter size
* 0 0 0: Internal calculations are Original size, output Original size
* 0 0 -1: Internal calculations are Original size, output Double size
*/
```

void KLDeallocate(Globals **glob);

/* Klics Function Definitions */

```
extern int KlicsEncode(short *src[3], short *dst[3], KlicsE kle);
extern Boolean KlicsDecode(short *src[3], short *dst[3], KlicsSeqHeader *seqh, Klics
    long mode, long scale[3], unsigned long *tree, unsigned long *wave);
```

```

.....
*
* Memory allocation/deallocation routines
*
*.....
```

```
OSErr
MemoryError()
```

```
{
    OSErr theErr;
```

```
#ifdef DEBUG
    if (0!==(theErr=MemError()))
        DebugStr("\pMemoryError");
```


- 688 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

#endif
    return(theErr);
}

OSErr
FreePtr(Ptr *ptr)
{
    OSErr theErr=0;

    if (*ptr!=nil) {
        DisposePtr(*ptr);
        *ptr=nil;
        theErr=MemoryError();
    }
    return(theErr);
}

#define FreePointer(handle,err) \
    if (noErr!=(err=FreePtr((Ptr*)&handle))) return(err)

extern OSErr Colour8(Ptr *);
extern OSErr Colour16(Ptr *);
extern OSErr UV32Table(Ptr *);
extern OSErr RGBTable(Ptr *);

OSErr
KLGetTab(Globals **glob,long new)
{
    OSErr theErr=0;
    SharedGlobals *sGlob=(*glob)->sharedGlob;
    long old=(*glob)->using;

    if (old!=new) {
        if (old!=None) {
            sGlob->use[old]--;
            if (sGlob->use[old]==0) {
                FreePointer(sGlob->tab[old],theErr);
            }
        }

        if (new!=None) {
            if (sGlob->use[new]==0)
                switch(new) {
                    #ifndef ENCODER
                    case Use8:
                        if (noErr!=(theErr=Colour8(&sGlob->tab[new])))
                            return(theErr);
                        break;
                    case Use16:
                        if (noErr!=(theErr=Colour16(&sGlob->tab[new])))
                            return(theErr);
                        break;
                    case Use32:
                        if (noErr!=(theErr=UV32Table(&sGlob->tab[new])))
                            return(theErr);
                        break;
                    #endif
                    #ifndef DECODER
                    case UseF32:
                        if (noErr!=(theErr=RGBTable(&sGlob->tab[new])))
                            return(theErr);
                        break;
                    #endif
                }
        }
    }
}

```

- 689 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

#endif
    )
    (*glob)->using=new;
    sGlob->use[new-1]++;
}

return(theErr);

OSErr
KLFree(Globals **glob)
{
    OSErr theErr=0;

    FreePointer((*glob)->src[0],theErr);
    FreePointer((*glob)->dst[0],theErr);
    FreePointer((*glob)->pixmap,theErr);
    (*glob)->size=0;
    return(theErr);
}

#define NewPointer(ptr,type,size) \
    saveZone=GetZone(); \
    SetZone(SystemZone()); \
    if (nil==(ptr=(type)NewPtr(size))) { \
        SetZone(ApplicZone()); \
        if (nil==(ptr=(type)NewPtr(size))) { \
            SetZone(saveZone); \
            return(MemoryError()); \
        } \
    } \
    SetZone(saveZone);

ComponentResult
KLMalloc(Globals **glob, short height, short width, long pixelSize)
{
    long    ysize,uvsize;
    THz     saveZone;

    ysize=(long)height * (long)width * (long)sizeof(short);
    uvsize = ysize>>2;

    if ((*glob)->size != ysize) {
        KLFree(glob);
        (*glob)->size = ysize;
        (*glob)->prev_frame=-1; /* frame doesn't contain valid data */

        /* Keep Src and Dst separate because of their large sizes */

        ysize=(long)height * (long)width * (long)sizeof(short) >> 2*(*glob)->scale
        uvsize = ysize>>2;
        NewPointer((*glob)->src[0],short *,ysize-uvsize+uvsize+16);
        (*glob)->src[1] = (short *)(((long)(*glob)->src[0] + ysize + 3L) & 0xFFFFF);
        (*glob)->src[2] = (short *)(((long)(*glob)->src[1] + uvsize + 3L) & 0xFFFFF);

        ysize=(long)height * (long)width * (long)sizeof(short) >> 2*(*glob)->scale
        uvsize = ysize>>2;
        NewPointer((*glob)->dst[0],short *,ysize+uvsize+uvsize+16);
        (*glob)->dst[1] = (short *)(((long)(*glob)->dst[0] + ysize + 3L) & 0xFFFFF);
        (*glob)->dst[2] = (short *)(((long)(*glob)->dst[1] + uvsize + 3L) & 0xFFFFF);
    }
}

```

- 690 -

Engineering:KlicsCode:CompPict:KlicsCDec.c

```

        NewPointer((*glob)->pixmap.Ptr,pixelSize/8*height*width<<1);
    }
    return(noErr);
}

OSErr
ResourceError()
{
    OSErr theErr;

#ifdef DEBUG
    if (0!==(theErr=ResError()))
        DebugStr("\pResourceError");
#endif
    return(theErr);
}

#ifdef COMPONENT
#define ResErr(resfile,err) \
    if (0!==(err=ResourceError())) { \
        if (resfile!=0) CloseComponentResFile(resfile); \
        return(err); \
    }
#else
#define ResErr(resfile,err) \
    if (0!==(err=ResourceError())) { \
        return(err); \
    }
#endif

ComponentResult
KLOpenInfoRes(ComponentInstance self, Handle *info)
{
    #pragma unused(self)
    short resFile=0;
    OSErr theErr=noErr;

    if (*info) {
        DisposHandle(*info);
        *info=nil;
    }
#ifdef COMPONENT
    resFile=OpenComponentResFile((Component)self);
    ResErr(resFile,theErr);
#else
    UseResFile(gResRef);
#endif
    *info=Get1Resource(codecInfoResourceType,128);
    *info=Get1Resource(codecInfoResourceType,129);
    ResErr(resFile,theErr);
    LoadResource(*info);
    ResErr(resFile,theErr);
    DetachResource(*info);
#ifdef COMPONENT
    CloseComponentResFile(resFile);
#endif
    return(theErr);
}

pascal ComponentResult
KLOpenCodec(ComponentInstance self)
{
    Globals **glob;

```

- 691 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

SharedGlobals *sGlob;
THz saveZone;
Boolean inAppHeap;
ComponentResult result = noErr;
short resFile=CurResFile();

DebugMsg("\pOpen Codec - begin");
if ( (glob = (Globals **)NewHandleClear(sizeof(Globals))) == nil ) {
    return(MemoryError());
} else HNoPurge((Handle)glob);
SetComponentInstanceStorage(self, (Handle)glob);

saveZone = GetZone();
inAppHeap = ( GetComponentInstanceA5(self) != 0 );
if ( !inAppHeap )
    SetZone(SystemZone());
if ( (sGlob=(SharedGlobals*)GetComponentRefcon((Component)self)) == nil ) {
    if ( (sGlob = (SharedGlobals*)NewPtrClear(sizeof(SharedGlobals))) == nil )
        result=MemoryError();
    goto obail;
}
SetComponentRefcon((Component)self, (long)sGlob);

(*glob)->sharedGlob = sGlob; // keep this around where it's easy to get at
if ( sGlob->info == nil || (*Handle)sGlob->info == nil ) {
    result=KLOpenInfoRes(self, &(Handle)(sGlob->info));
    HNoPurge((Handle)sGlob->info);
}

obail:
SetZone(saveZone);
if ( result != noErr && sGlob != nil ) {
    if ( sGlob->info )
        DisposHandle((Handle)sGlob->info);
    DisposPtr((Ptr)sGlob);
    SetComponentRefcon((Component)self, (long)nil);
}
(*glob)->size=0;
DebugMsg("\pOpen Codec - end");
return(result);
}

pascal ComponentResult
KLCloseCodec(Handle storage, ComponentInstance self)
{
    SharedGlobals *sGlob;
    Globals **glob = (Globals **)storage;

    DebugMsg("\pClose Codec - begin");
    HLock(storage);
    if ( glob ) {
        KLFree(glob);
        KLGetTab(glob, None);
        if (CountComponentInstances((Component)self) == 1) {
            if ( (sGlob=(SharedGlobals*)(*glob)->sharedGlob) != nil ) {
                if ( sGlob->info )
                    HPurge((Handle)sGlob->info);
            }
        }
        DisposHandle((Handle)glob);
    }
}

```

- 692 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

    height = 120;
}
if (time)
    *time = (width * height * 11);

if (*spatialQuality && *spatialQuality==codecLosslessQuality)
    *spatialQuality = codecMaxQuality;

if (*temporalQuality && *temporalQuality==codecLosslessQuality)
    *temporalQuality = codecMaxQuality;

return(noErr);
}
/*
 * Extends dimensions to make a multiples of 32x16
 */

#define KLExtendWidth(dim) 31-(dim-1&31)
#define KLExtendHeight(dim) 15-(dim-1&15)

pascal ComponentResult
KLGetMaxCompressionSize(Handle storage, PixMapHandle src, const Rect *srcRect, short
    CodecQ quality, long *size)
{
    *pragma unused(storage,src,depth,quality);
    short width = srcRect->right - srcRect->left;
    short height = srcRect->bottom - srcRect->top;

    /* test by just doing RGB storage */

    *size = 3 * (width+KLExtendWidth(width)) * (height+KLExtendHeight(height));
    return(noErr);
}

pascal ComponentResult
KLGetCompressedImageSize(Handle storage, ImageDescriptionHandle desc, Ptr data, long
    DataProcRecordPtr dataProc, long *size)
{
    *pragma unused(storage,dataSize,dataProc,desc);
    short frmh_size;
    long data_size;

    if ( size == nil ) {
        return(paramErr);
    }
    frmh_size=((KlicsHeader *)data)->description_length;
    data_size=((KlicsFrameHeader *)data)->length;
    *size=(long)frmh_size+data_size;
    return(noErr);
}

void KLSaveSetup(Boolean still, short width, short height, CodecQ space, CodecQ tem
{
    kle->seqh.head.description_length=sizeof(KlicsSeqHeader);
    kle->seqh.head.version_number[0]=0;
    kle->seqh.head.version_number[1]=1;
    kle->seqh.sequence_size[0]=width;
    kle->seqh.sequence_size[1]=height;
    kle->seqh.sequence_size[2]=0;
    kle->seqh.sub_sample[0]=1;
    kle->seqh.sub_sample[1]=1;
    kle->seqh.wavelet=WT_Daub4;
}

```

- 693 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

kle->seqh.precision=10;
kle->seqh.octaves[0]=3;
kle->seqh.octaves[1]=2;

kle->fmh.head.description_length=sizeof(KlicsFrameHeader);
kle->fmh.head.version_number[0]=0;
kle->fmh.head.version_number[1]=1;

kle->encd.bpf_in=(2133+temp*160)/8; /* High = 64000 bits/frame, Poor = 1
kle->encd.bpf_out=kle->encd.bpf_in;
kle->encd.buf_size=kle->encd.bpf_in*4;

kle->encd.quant=16-(space*15)/1023;
kle->encd.thresh=1.0;
kle->encd.compare=1.0;
kle->encd.base[0]=0.10;
kle->encd.base[1]=0.10;
kle->encd.base[2]=0.20;
kle->encd.base[3]=0.50;
kle->encd.base[4]=1.00;
kle->encd.intra=still;
kle->encd.auto_q=true;
kle->encd.buf_sw=true;
kle->encd.prevquact=1;
kle->encd.prevbytes=13;
}

#ifndef DECODER
pascal ComponentResult
KLPreCompress(Handle storage,register CodecCompressParams *p)
{
    ComponentResult result;
    CodecCapabilities *capabilities = p->capabilities;
    short width=(*p->imageDescription)->width+(capabilities->extendW
    short height=(*p->imageDescription)->height+(capabilities->exten
    Globals **glob=(Globals **)storage;
    KlicsE kle=&(*glob)->kle;
    Handle ext=NewHandle(sizeof(KlicsSeqHeader));

    DebugMsg("\pKLPreCompress");
    HLock(storage);
    if (MemError()!=noErr) return(MemError());
    switch ( (*p->imageDescription)->depth ) {
        case 24:
            capabilities->wantedPixelSize = 32;
            kle->seqh.channels=3;
            if (noErr!=(result=KLGetTab(glob,UseF32)))
                return(result);
            break;
        default:
            return(codecConditionErr);
            break;
    }

    /* Going to use 3 octaves for Y and 2 for UV so the image must be a multiple o
    capabilities->bandMin = height;
    capabilities->bandInc = capabilities->bandMin;

    capabilities->flags=codecCanCopyPrevComp|codecCanCopyPrev;

    (*glob)->scale[0]=0;
    (*glob)->scale[1]=0;

```

- 694 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

(*glob)->scale[2]=0;

if (noErr!=(result=KLMalloc(glob,height,width,0))) return result;
KLSetup(p->sequenceID==0,width,height,(*p->imageDescription)->spatialQuality,1

BlockMove((Ptr)&kle->seqh,*ext,sizeof(KlicsSeqHeader));
if (noErr!=(result=SetImageDescriptionExtension(p->imageDescription,ext,klicsC
return result;

HUnlock(storage);
DebugMsg("\pKLPreCompress success");
return(result);
)
#endif

#ifdef ENCODER
pascal long
KLPreDecompress(Handle storage,register CodecDecompressParams *p)
(
ComponentResult      result;
CodecCapabilities    *capabilities = p->capabilities;
Rect                 dRect = p->srcRect;
long                 width;
long                 height;
long                 channels;
Glcbls               **glob=(Glcbls **)storage;
KlicsE               kle;
Handle               ext;
OSErr                err;

DebugMsg("\pKLPreDecompress");
if (!TransformRect(p->matrix,&dRect,nil) )
    return(codecConditionErr);

HLock(storage);
kle=&(*glob)->kle;
switch ( (*p->imageDescription)->depth ) (
    case 24:
        switch(p->dstPixmap.pixelSize) (
            case 32:
                capabilities->wantedPixelSize = 32;
                if (p->conditionFlags&codecConditionNewDepth) {
                    if (noErr!=(err=KLGetTab(glob,Use32)))
                        return(err);
                }
                break;
            case 16:
                capabilities->wantedPixelSize = 16;
                if (p->conditionFlags&codecConditionNewDepth) {
                    if (noErr!=(err=KLGetTab(glob,Use16)))
                        return(err);
                }
                break;
            case 8:
                capabilities->wantedPixelSize = 8;
                if (p->conditionFlags&codecConditionNewClut) {
                    if (noErr!=(err=KLGetTab(glob,Use8)))
                        return(err);
                }
                break;
        )
        channels=3;
        break;

```

- 695 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

default:
    return(codecConditionErr);
    break;
)

if (noErr!=(result=GetImageDescriptionExtension(p->imageDescription,&ext,klics.
BlockMove(&ext,(Ptr)&kle->seqh,sizeof(KlicsSeqHeader)));
if (channels==1) kle->seqh.channels=1;

/* Going to use 3 octaves for Y and 2 for UV so the image must be a multiple o

#ifdef HQ
(*glob)->scale[0]=0; /* Tree scale */
#else
(*glob)->scale[0]=1; /* Tree scale */
#endif
width=kle->seqh.sequence_size[0];
height=kle->seqh.sequence_size[1];

switch((*glob)->scale[0]) {
case 1: /* Quarter size internal */
    (*glob)->scale[1]=1;
    if (p->matrix->matrix[0][0]==p->matrix->matrix[1][1])
        switch(p->matrix->matrix[0][0]) {
        case 32768:
            capabilities->flags=codecCanScale;
            capabilities->extendWidth=width/2-dRect.right;
            capabilities->extendHeight=height/2-dRect.bottom;
            (*glob)->scale[2]=1;
            break;
        case 65536:
            capabilities->extendWidth=width-dRect.right;
            capabilities->extendHeight=height-dRect.bottom;
            (*glob)->scale[2]=0;
            break;
        default:
            capabilities->extendWidth=0;
            capabilities->extendHeight=0;
            (*glob)->scale[2]=0;
            break;
        }
    else {
        capabilities->extendWidth=0;
        capabilities->extendHeight=0;
        (*glob)->scale[2]=0;
    }
    break;
case 0: /* Full size internal */
    if (p->matrix->matrix[0][0]==p->matrix->matrix[1][1])
        switch(p->matrix->matrix[0][0]) {
        case 32768:
            capabilities->flags=codecCanScale;
            capabilities->extendWidth=width/2-dRect.right;
            capabilities->extendHeight=height/2-dRect.bottom;
            (*glob)->scale[1]=1;
            (*glob)->scale[2]=1;
            break;
        case 131072:
            capabilities->flags=codecCanScale;
            capabilities->extendWidth=width*2-dRect.right;
            capabilities->extendHeight=height*2-dRect.bottom;
            (*glob)->scale[1]=0;
            (*glob)->scale[2]=-1;

```


- 696 -

Engineering:KlicsCode:CompFact:KlicsCodec.c

```

        break;
    case 65536:
        capabilities->extendWidth=width-dRect.right;
        capabilities->extendHeight=height-dRect.bottom;
        (*glob)->scale[1]=0;
        (*glob)->scale[2]=0;
        break;
    default:
        capabilities->extendWidth=0;
        capabilities->extendHeight=0;
        (*glob)->scale[1]=0;
        (*glob)->scale[2]=0;
    }
    else {
        capabilities->extendWidth=0;
        capabilities->extendHeight=0;
        (*glob)->scale[1]=0;
        (*glob)->scale[2]=0;
    }
    break;
}

capabilities->bandMin = height;
capabilities->bandInc = capabilities->bandMin;
capabilities->flags|=codecCanCopyPrev|codecCanCopyPrevComp|codecCanRemapColor;

if (noErr!=(result=KLMalloc(glob,height,width,capabilities->wantedPixelSize)))

HUnlock(storage);
DebugMsg("\pKLPreDecompress success");
return(result);
}
#endif

/* Test Versions in C - Colour.c */
void RGB2YUV32(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int wid
void YUV2RGB32(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int wid
void YUV2RGB32x2(Ptr table, long *pixmap, short *Yc, short *Uc, short *Vc, int a

/* Assembler versions - Colour.a */
OUT32X2(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, l
OUT32X2D(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, l
OUT32(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon
OUT32D(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon
OUT8X2(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon
OUT8(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon
OUT16X2(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, l
OUT16(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon
IN32(Ptr table, long *pixmap, short *Y, short *U, short *V, long width, long height, lon

/* Assembler versions - Color2.a */
void RGB2YUV2(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int widt
void YUV2RGB2(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int widt
void YUV2RGB3(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int widt
void GREY2Y(long *pixmap, short *Yc, int area, int width, int cols);
void Y2GREY(long *pixmap, short *Yc, int lines, int width, int cols);
void Y2GGG(long *pixmap, short *Yc, int lines, int width, int cols);

/*YUV2RGB4((*glob)->Table,pixmap,src[0],src[1],src[2],cols*(*desc)->height>>scale,
YUV2RGB5((*glob)->Table,pixmap,src[0],src[1],src[2],cols*(*desc)->height,width>>sc

#pragma parameter __D0 MicroSeconds

```

- 697 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```
pascal unsigned long MicroSeconds(void) = (0x4EB0, 0x81E1, 0x64C);
```

```
unsigned long GetTimerValue(unsigned long *TimerRes)
```

```
{
    *TimerRes = CLOCKS_PER_SEC;
    return(MicroSeconds());
}
```

```
#ifndef DECODER
```

```
pascal long
```

```
KLBandCompress(Handle storage,register CodecCompressParams *p)
```

```
{
#pragma unused(storage)
    Globals **glob = (Globals **)storage;
    ImageDescription **desc = p->imageDescription;
    char *baseAddr;
    short rowBytes;
    Rect sRect;
    long offsetH,offsetV;
    OSErr result = noErr;
    short *src[3],*dst[3];
    long *pixmap;
    int width=(*desc)->width+KLExtendWidth((*desc)->width);
    int height=(*desc)->height+KLExtendHeight((*desc)->height);
    int hwidth=width>>1,hheight=height>>1;
    int bytes;
    KlicsE kle;
    char mmuMode=1;
    char intra[]="\pENC:Intra-mode", inter[]="\pENC:Inter-mode";
    SharedGlobals *sGlob;
```

```
#ifdef PERFORMANCE
```

```
(void)PerfControl(ThePGlobals,true);
```

```
#endif
```

```
    DebugMsg("\pBandCompress");
    HLock((Handle)glob);
    kle=(*glob)->kle;
    sGlob=(*glob)->sharedGlob;
```

```
    rowBytes = p->srcPixmap.rowBytes & 0x3fff;
```

```
    sRect = p->srcPixmap.bounds;
```

```
    switch ( p->srcPixmap.pixelSize ) {
```

```
    case 32:
        offsetH = sRect.left<<2;
        break;
```

```
    case 16:
        offsetH = sRect.left<<1;
        break;
```

```
    case 8:
        offsetH = sRect.left;
        break;
```

```
    default:
        result = codecErr;
        DebugMsg("\pError");
        goto bail;
```

```
    }
    offsetV = sRect.top * rowBytes;
```

```
    baseAddr = p->srcPixmap.baseAddr + offsetH + offsetV;
```

```
    pixmap=(long *)baseAddr;
```

```
/* F5MakeF5Spec(0.0,"\pUser:crap001",&f5spec);
```

```
F5pCreate(&f5spec,"????","????",-1);
```

- 698 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

FSOpenDF(&fsspec, fswrPerm, &fileRefNum);
area=height*rowBytes;
FWrite(fileRefNum, &area, (long*)pixmap);
FSClose(fileRefNum);

src[0]=(*glob)->src[0]; src[1]=(*glob)->src[1]; src[2]=(*glob)->src[2];
dst[0]=(*glob)->dst[0]; dst[1]=(*glob)->dst[1]; dst[2]=(*glob)->dst[2];
switch(kle->seqn.channels) {
case 3:
    IN32(&sclob->tab[UseF32-1], pixmap, src[0], src[1], src[2], width, height, rowByte
    break;
}

/.....
* Klics encode
/.....
#ifdef DEBUG
if (p->callerFlags&codecFlagUseImageBuffer) DebugStr("\pUseImageBuffer"); /*
if (p->callerFlags&codecFlagUseScreenBuffer) DebugStr("\pUseScreenBuffer"); /*
if (p->callerFlags&codecFlagUpdatePrevious) DebugStr("\pUpdatePrevious"); /*
if (p->callerFlags&codecFlagNoScreenUpdate) DebugStr("\pNoScreenUpdate"); /*
if (p->callerFlags&codecFlagDontOffscreen) DebugStr("\pDontOffscreen"); /*
if (p->callerFlags&codecFlagUpdatePreviousComp) DebugStr("\pUpdatePreviousComp
if (p->callerFlags&codecFlagForceKeyFrame) DebugStr("\pForceKeyFrame"); /*
if (p->callerFlags&codecFlagOnlyScreenUpdate) DebugStr("\pOnlyScreenUpdate");
#endif

kle->buf.buf=(unsigned long *) (p->data+sizeof(KlicsFrameHeader));
kle->encl.intra=(p->temporalQuality==0);
kle->frmh.frame_number=p->frameNumber;

bytes=KlicsEncode(src, dst, kle);

BlockMove((Ptr)&kle->frmh, p->data, sizeof(KlicsFrameHeader));
bytes+=sizeof(KlicsFrameHeader);

(*glob)->prev_frame=p->frameNumber;

p->data+=bytes;
p->bufferSize=bytes;
(*p->imageDescription)->dataSize=bytes;

p->similarity=(kle->encl.intra?0:Long2Fix(244));
p->callerFlags=0;
/* p->callerFlags|=codecFlagUsedImageBuffer|(kle->encl.intra?codecFlagUsedNewImage
bail:
    HUnlock((Handle)glob);
#ifdef PERFORMANCE
if (0!=(result=PerfDump(ThePGlobals, "\pEncode.perf", false, 0)))
    return(result);
#endif
    DebugMsg("\pBandCompress success");
    return(result);
}
#endif

/* Display stuff for debugging
CGrafPtr wPort, savePort;

```

- 699 -

Engineering:KlicsCode:CcmpPict:KlicsCodec.c

```

Rect      rect;
Str255    str;

GetPort((GrafPtr *)&savePort);
GetCWMgrPort(&wPort);
SetPort((GrafPtr)wPort);
SetRect(&rect, 0, 0, 50, 30);
ClipRect(&rect);
EraseRect(&rect);
NumToString(frmh->frame_number, str);
MoveTo(0, 20);
DrawString(str);
if (frmh->flags & KFH_INTRA) {
    SetRect(&rect, 0, 30, 50, 65);
    ClipRect(&rect);
    EraseRect(&rect);
    NumToString(frmh->frame_number/24, str);
    MoveTo(0, 50);
    DrawString(str);
}
SetRect(&rect, -2000, 0, 2000, 2000);
ClipRect(&rect);
SetPort((GrafPtr)savePort); */

#define flag_tree    0x1
#define flag_wave    0x2
#define flag_show    0x4
#define flag_full    0x8
#define DURATION     65666

long  ModeSwitch(Globals *glob, KlicsFrameHeader *frmh)
{
    long  mode=0, i, fps;
    Boolean repeat=glob->prev_frame==frmh->frame_number,
           next=glob->prev_frame+1==frmh->frame_number;
    CGrafPtr  wPort, savePort;
    Rect      rect;
    Str255    str;

    DebugMsg('\pModeSwitch - begin');
    if (frmh->frame_number==0)
        for(i=0; i<15; i++) glob->out[i]=false;
    if (repeat) {
        glob->run_time=0;
        DebugMsg('\pModeSwitch - repeat (end)');
        return(flag_show|flag_full);
    }

    if (next)
        switch(frmh->flags) {
            case KFH_SKIP:
                DebugMsg('\pModeSwitch - next/skip');
                glob->prev_frame=frmh->frame_number;
                if (glob->sys_time>DURATION) {
                    glob->run_time=0;
                    if (glob->real_frame!=glob->dpy_frame)
                        mode|=flag_wave|flag_show;
                } else {
                    unsigned long frame, late;

                    frame=glob->run_frame+(glob->sync_time-glob->run_time)/DURATION;
                    late=(glob->sync_time-glob->run_time)%DURATION;
                    if (frame<=glob->prev_frame && glob->real_frame!=glob->dpy_frame)

```

- 700 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

    mode=flag_wave|flag_show;
    if (frame<=glob->prev_frame && late+glob->wave_time+glob->spy_time
        mode=flag_wave|flag_show;*/
    }
    break;
case KFH_INTRA:
    DebugMsg("\pModeSwitch - next/intra");
    mode=flag_tree;
    glob->prev_frame=frmh->frame_number;
    glob->real_frame=glob->prev_frame;
    if (glob->sys_time>DURATION) {
        glob->run_time=0;
        mode=flag_wave|flag_show|flag_full;
    } else
/*      if (glob->run_time==0) {*/
        glob->key_time=glob->sync_time-glob->run_time;
        glob->run_time=glob->sync_time-glob->sys_time;
        glob->run_frame=glob->prev_frame;
        mode=flag_wave|flag_show|flag_full;
/*      } else {
        unsigned long frame, late;

        frame=glob->run_frame+(glob->sync_time-glob->run_time)/DURATION;
        late=(glob->sync_time-glob->run_time)%DURATION;
        if (frame<=glob->prev_frame)
            mode=flag_wave|flag_show|flag_full;
        }*/
    break;
default:
    DebugMsg("\pModeSwitch - next/inter");
    mode=flag_tree;
    glob->prev_frame=frmh->frame_number;
    glob->real_frame=glob->prev_frame;
    if (glob->sys_time>DURATION) {
        glob->run_time=0;
        mode=flag_wave|flag_show;
    } else
        if (glob->run_time==0) {
            glob->run_time=glob->sync_time-glob->sys_time;
            glob->run_frame=glob->prev_frame;
            mode=flag_wave|flag_show;
        } else {
            unsigned long frame, late;

            frame=glob->run_frame+(glob->sync_time-glob->run_time)/DURATION;
            late=(glob->sync_time-glob->run_time)%DURATION;
            if (frame<=glob->prev_frame)
                mode=flag_wave|flag_show;
            if (frame<=glob->prev_frame && late+glob->tree_time+glob->wave
                mode=flag_wave|flag_show;*/
        }
    break;
}
else
    switch(frmh->flags) {
    case KFH_SKIP:
        DebugMsg("\pModeSwitch - jump/skip");
        glob->run_time=0;
        break;
    case KFH_INTRA:
        DebugMsg("\pModeSwitch - jump/intra");
        mode=flag_tree|flag_wave|flag_show|flag_full;
        for(i=glob->prev_frame;i<frmh->frame_number;i++)

```

- 701 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

    glob->out[frmh->frame_number%15]=0;
    glob->prev_frame=frmh->frame_number;
    glob->real_frame=glob->prev_frame;
    glob->run_time=0;
    break;
default:
    DebugMsg("\pModeSwitch - jump/inter");
    glob->run_time=0;
    break;
}
DebugMsg("\pModeSwitch - display info");
#ifdef COMPONENT
/* glob->out[frmh->frame_number%15]=(mode&flag_show)!=0;
for(i=0;fps=0;i<15;i++) if (glob->out[i]) fps++;
GetPort((GrafPtr *)&savePort);
GetCWMgrPort(&wPort);
SetPort((GrafPtr)wPort);
SetRect(&rect,0,20,120,50);
ClipRect(&rect);
EraseRect(&rect);
NumToString(frmh->frame_number,str);
MoveTo(0,35);
DrawString(str);
DrawString("\p:");
NumToString(fps,str);
DrawString(str);
MoveTo(0,50);
for(i=0;i<15;i++)
    if (glob->out[i]) DrawString("\pX");
    else DrawString("\pO");
SetRect(&rect,-2000,0,2000,2000);
ClipRect(&rect);
SetPort((GrafPtr)savePort);*/
#endif
DebugMsg("\pModeSwitch - end");
return(mode);
}

#ifdef ENCODER
pascal long
KLBandDecompress(Handle storage,register CodecDecompressParams *p)
{
#pragma unused(storage)
Globals **glob = (Globals **)storage;
ImageDescription **desc = p->imageDescription;
int x,y;
char *baseAddr;
short rowBytes;
Rect dRect;
long offsetH,offsetV;
OSErr result = noErr;
short *src[3],*dst[3];
long *pixmap;
int width=(*desc)->width+KLExtendWidth((*desc)->width);
int height=(*desc)->height+KLExtendHeight((*desc)->height);
int bwidth=width>>1,hheight=height>>1,area=height*width;
Klics2 Klc;
KlicsFrameHeader *frmh;
char mmuMode=1;
long mode;
SharedGlobals *sGlob;
/*
FILE *fp;

```

- 702 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

char      file_name[30];
CGrafPtr  wPort, savePort;
Rect      rect;
Str255    str;

/*

HLock((Handle)glob);
DebugMsg("\pBandDecompress");
(*glob)->sys_time=GetTimeValue(&(*glob)->sys_time);
(*glob)->sys_time-=(*glob)->sync_time;

#ifdef PERFORMANCE
(void)PerfControl(ThePGlobals,true);
#endif

kle=(*glob)->kle;
sGlob=(*glob)->sharedGlob;

dRect = p->srcRect;
if ( !TransformRect(p->matrix,&dRect,nil) ) {
    DebugMsg("\pTransformRect Error");
    return(paramErr);
}
rowBytes = p->dstPixMap.rowBytes & 0x3fff;
offsetH = (dRect.left - p->dstPixMap.bounds.left);
switch ( p->dstPixMap.pixelSize ) {
case 32:
    offsetH <<=2;
    break;
case 16:
    offsetH <<=1;
    break;
case 8:
    break;
default:
    result = codecErr;
    DebugMsg("\pDepth Error");
    goto bail;
}
offsetV = (dRect.top - p->dstPixMap.bounds.top) * rowBytes;
baseAddr = p->dstPixMap.baseAddr + offsetH + offsetV;
pixmap=(long *)baseAddr;

/*****
 *
 *   Klics decode
 *
 *****/

src[0]=(*glob)->src[0]; src[1]=(*glob)->src[1]; src[2]=(*glob)->src[2];
dst[0]=(*glob)->dst[0]; dst[1]=(*glob)->dst[1]; dst[2]=(*glob)->dst[2];

frmh=(KlicsFrameHeader *)p->data;
kle->buf.buf=(unsigned long *) (p->data+sizeof(KlicsFrameHeader));
mode=ModeSwitch(*glob,frmh);

KlicsDecode(src,dst,&kle->seqh,frmh,&kle->buf,mode,(*glob)->scale,&(*glob)->tr

if ( kle->buf.ptr-kle->buf.buf > frmh->length+2)
    DebugMsg("\pWarning: Decompressor read passed end of buffer");

p->data[0]='X';
p->data[1]=mode&flag_tree?'T':' ';

```

- 703 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```

p->data[2]=mode&flag_wave?'W':0;
p->data[3]=mode&flag_show?'S':0;
p->data[4]=mode&flag_full?'F':0;
p->data[5]=frmh->flags&KFM_INTRA?'I':0;
p->data[6]=frmh->flags&KFM_SKIP?'K':0;
p->data[7]='X';

p->data+=p->bufferSize;

/.....
*   signed 10 bit YUV-unsigned 8 RGB convert
*...../

#ifdef COMPONENT
    SwapMMUMode(&mmuMode);
#endif
if (mode&flag_show) {
    (*glob)->sync_time=GetTimerValue(&(*glob)->sync_time);
    (*glob)->dpy_frame=(*glob)->real_frame;
    if ((*glob)->scale[2]<(*glob)->scale[1]) {
        switch(kle->seqh.channels) {
            case 3:
                switch (p->dstPixMap.pixelSize) {
                    case 32:
                        if (mode&flag_full)
                            OUT32X2(sGlob->tab[Use32-1],pixmap,src[0],src[1],src[2],wi.
                        else
                            OUT32X2D(sGlob->tab[Use32-1],pixmap,src[0],src[1],src[2],w
                        break;
                    case 16:
                        OUT16X2(sGlob->tab[Use16-1],pixmap,src[0],src[1],src[2],width>
                        break;
                    case 8:
                        OUT8X2(sGlob->tab[Use8-1],pixmap,src[0],src[1],src[2],width>>
                        break;
                }
                break;
            ..
        }
    } else {
        switch(kle->seqh.channels) {
            case 3:
                switch (p->dstPixMap.pixelSize) {
                    case 32:
                        if (mode&flag_full)
                            OUT32(sGlob->tab[Use32-1],pixmap,src[0],src[1],src[2],width
                        else
                            OUT32D(sGlob->tab[Use32-1],pixmap,src[0],src[1],src[2],wid
                        break;
                    case 16:
                        OUT16(sGlob->tab[Use16-1],pixmap,src[0],src[1],src[2],width>>
                        break;
                    case 8:
                        OUT8(sGlob->tab[Use8-1],pixmap,src[0],src[1],src[2],width>>(*g
                        break;
                }
                break;
        }
    }
    (*glob)->dpy_time=GetTimerValue(&(*glob)->dpy_time);
    (*glob)->dpy_time-=(*glob)->sync_time;
}

```


- 704 -

Engineering:KlicsCode:CompPict:KlicsCodec.c

```
CLEARA2();
(*glob) -> sync_time = GetTimerValue(&(*glob) -> sync_time);

#ifdef COMPONENT
    SwapMMUMode(&mmuMode);
#endif

fail:
    HUnlock(Handle(glob));

#ifdef PERFORMANCE
    if(0 != (result = PerfDump(ThePGlobals, "\pDeccode.perf", false, 0)))
        return(result);
#endif
    DebugMsg("\pBandDecompress success");
    return(result);
}
#endif
```

- 705 -

Engineering:KlicsCode:CompPict:Klics.h

```

.....
.
.  © Copyright 1993 KLICS Limited
.  All rights reserved.
.
.  Written by: Adrian Lewis
.
...../
/*
.  Second generation header file
*/

#include    <stdio.h>

/* useful X definitions */
/*typedef char    Boolean;*/
typedef char    *String;
#define True    1
#define False    0

/* new Blk definition */
typedef int    Blk[4];

#define WT_Haar    0
#define WT_Daub4    1

/* mode constructors */
#define M_LPF    1
#define M_STILL    2
#define M_SEND    4
#define M_STOP    8
#define M_VOID    16
#define M_QUIT    32

/* LookAhead histogram */
#define HISTO    300
#define HISTO_DELTA    15.0
#define HISTO_BITS    10

/* Fast Functions */

/* Is the block all zero ? */
#define BlkZero(block) \
    block[0]==0 && block[1]==0 && block[2]==0 && block[3]==0

/* Sum of the absolute values */
#define Decide(new) \
    abs(new[0])+ \
    abs(new[1])+ \
    abs(new[2])+ \
    abs(new[3])

/* Sum of the absolute differences */
#define DecideDelta(new,old) \
    abs(new[0]-old[0])+ \
    abs(new[1]-old[1])+ \
    abs(new[2]-old[2])+ \
    abs(new[3]-old[3])

/* Adjust the norm for comparison with SigmaAbs */
#define DecideDouble(norm) (4.0*norm)

/* Get addresses from x,y coords of block, sub-band, octave.

```

- 706 -

Engineering:KlitsCode:CompPict:Klits.h

```

/* image size and mask (directly related to octave) information */
#define GetAddr(addr,x,y,sub,oct,size,mask) \
{ int    smask=mask>>1; \
  x0=x!(sub&1?smask:0); \ \
  x1=x!(sub&1?smask:0)!mask; \
  y0=y!(sub&2?smask:0)*size[0]; \
  y1=y!(sub&2?smask:0)!mask)*size[0]; \
  addr[0]=x0-y0; \
  addr[1]=x1-y0; \
  addr[2]=x0+y1; \
  addr[3]=x1+y1; \
}

/* Get data values from addresses and memory */
#define GetData(addr,block,data) \
  block[0]=(int)data[addr[0]]; \
  block[1]=(int)data[addr[1]]; \
  block[2]=(int)data[addr[2]]; \
  block[3]=(int)data[addr[3]];

#define VerifyData(block,mask,tmp) \
  tmp=block&mask; \
  if (tmp!=0 && tmp!=mask) { \
    block=block<0?mask:-mask; \
  }

/* Put data values to memory using addresses */
#define PutData(addr,block,data) \
  data[addr[0]]=(short)block[0]; \
  data[addr[1]]=(short)block[1]; \
  data[addr[2]]=(short)block[2]; \
  data[addr[3]]=(short)block[3];

/* Put zero's to memory using addresses */
#define PutZero(addr,data) \
  data[addr[0]]=0; \
  data[addr[1]]=0; \
  data[addr[2]]=0; \
  data[addr[3]]=0;

/* Mode: M_VOID Put zero's and find new mode */
#define DoZero(addr,dst,mode,oct) \
  PutZero(addr,dst); \
  mode[oct]=oct==0?M_STOP:M_VOID;

/* Descend the tree structure
 * Copy mode, decrement octave (& mask), set branch to zero
 */
#define DownCounters(mode,oct,mask,blk) \
  mode[oct-1]=mode[oct]; \
  oct--; \
  mask = mask>>1; \
  blk[oct]=0;

/* Ascend the tree structure
 * Ascend tree (if possible) until branch not 3
 * If at top then set mode to M_QUIT
 * Else increment branch and x, y coords
 */
#define StopCounters(mode,oct,mask,blk,x,y,octs) \
  while(oct<octs-1 && blk[oct]!=3) { \

```

- 707 -

Engineering:KlicsCode:CompPict:Klics.h

```
    blk{oct}=0; \
    mask= mask<<1; \
    x ^= -mask; \
    y ^= -mask; \
    oct++; \
} \
if (oct==octx-1) mode{oct}=M_QUIT; \
else { \
    blk{oct}++; \
    x ^= mask<<1; \
    if (blk{oct}==2) y ^= mask<<1; \
    mode{oct}=mode{oct+1}; \
}
```

Engineering:KLICSCode:CompPict:Haar.a

```

-----
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
-----
*
*  68000 FastForward/Backward Haar
*
-----
*
*  macro
*  Fwd0      &addr0,&dG,&dH
*
*  move.w    (&addr0),&dG      ; dG=(short *)addr1
*  move.w    &dG,&dH           ; dH=dG
*
*  endm
*
-----
*
*  macro
*  Fwd1      &addr1,&addr0,&dG,&dH
*
*  move.w    (&addr1),d0      ; v=(short *)addr2
*  add.w     d0,&dH           ; dH+=v
*  sub.w     d0,&dG           ; dG-=v
*  clr.w     d0              ; d0=0
*  asr.w     #1,&dH          ; dAH>>=1
*  addx.w    d0,&dH           ; round dH
*  asr.w     #1,&dG          ; dG>>=1
*  addx.w    d0,&dG           ; round dG
*  move.w    &dH,(&addr0)    ; *(short *)addr0=dH
*  move.w    &dG,(&addr1)    ; *(short *)addr1=dG
*
*  mend
*
-----
*
*  macro
*  Fwd      &base,&end,&inc
*
*  movea.l   &base,a0        ; addr0=base
*  move.l    &inc,d0          ; d0=inc
*  asr.l     #1,d0           ; d0=inc>>1
*  movea.l   a0,a1           ; addr1=addr0
*  suba.l    d0,a1           ; addr1-=(inc>>1)
*  @do      Fwd0             ; Fwd0(addr0,dG,dH)
*  adda.l    &inc,a1         ; addr1+=inc
*  Fwd1      a1,a0,d4,d5     ; Fwd1(addr1,addr0,dG,dH)
*  adda.l    &inc,a0         ; addr0+=inc
*  cmpa.l    a0,&end         ; addr0<end
*  bgt.s     @do            ; while
*
*  endm
*
-----
*
*  HaarForward FUNC      EXPORT
*
*  link      a6,#0          ; no local variables
*  movem.l   d4-d7/a3-a5,-(a7) ; store registers
*
*  move.l    $000C(a6),d3    ; inc=inc1
*  movea.l   $0008(a6),a5    ; base=data
*  move.l    $0010(a6),d6    ; endl
*  move.l    $0018(a6),d7    ; end2
*  move.l    $0014(a6),d2    ; inc2

```

- 709 -

Engineering:KlicsCode:CompPict:Haar.a

```

@do    movea.l    a5,a4          : end=base
      adda.l     d6,a4          : end+=endl
      fvd        a5,a4,d3       : fvd(base,end,inc)
      adda.l     d2,a5          : base+=inc2
      cmpa.l     d7,a5          : end2>base
      blt.s      @do           : for

      movem.l    (a7)+,d4-d7/a3-a5 : restore registers
      unlk      a6             : remove locals
      rts                          : return

      ENDFUNC
-----
      macro
      Bwd0      &addr0,&dG,&dH

      move.w     (&addr0),&dG    ; dG=(short *)&addr0
      move.w     &dG,&dH         ; dH=dG

      endm

      macro
      Bwd1      &addr1,&addr0,&dG,&dH

      move.w     (&addr1),d0     ; v=(short *)&addr1
      add.w      d0,&dH          ; dH+=v
      sub.w      d0,&dG          ; dG-=v
      move.w     &dH,(&addr0);   ; *(short *)&addr0=dH
      move.w     &dG,(&addr1);   ; *(short *)&addr1=dG

      endm

      macro
      Bwd      &base,&count,&inc

      movea.l    &base,a0        ; addr0=base
      move.l     &inc,d0         ; d0=inc
      asr.l      #1,d0           ; d0=inc>>1
      movea.l    a0,a1           ; addr1=addr0
      suba.l     d0,a1           ; addr1=(inc>>1)*
      @do        a0,d4,d5        ; Bwd0(addr0,dG,dH)
      adda.l     &inc,a1         ; addr1+=inc
      Bwd1      a1,a0,d4,d5      ; Bwd1(addr1,addr0,dG,dH)
      adda.l     &inc,a0         ; addr0+=inc
      dbf        &count,@do     ; while --!!=count

      endm
-----
HaarBackward  FUNC  EXPORT
*   d0 - spare, d1 - count1, d2 - inc2, d3 - incl, d4 - dG, d5 - dH, d6 - loop1, d
*
      link      a6,#0           : no local variables
      movem.l   d4-d7/a3-a5,-(a7) : store registers

      move.l     $000C(a6),d3     ; inc=incl
      movea.l    $0008(a6),a5     ; base=data
      move.l     $0010(a6),d6     ; loop1 (width/height)
      move.l     $0018(a6),d7     ; loop2 (height/width)
      move.l     $0014(a6),d2     ; inc2
      subq.l     #1,d7            ; loop2-=1
      lsr.l      #1,d6           ; loop1/=2
      subq.l     #1,d6           ; loop1-=1

```

- 710 -

Engineering:KlicsCode:CompPict:Haar.a

```

ndo      move.l    d6,d1          ; count1=loop1
        bwd        a5,d1,d3      ; bwd(base.count,inc)
        adda.l     d2,a5          ; base+=inc2
        dbf        d7,edo        ; while -1!--loop2

        movem.l    (a7)-,d4-d7/a3-a5 ; restore registers
        unlk       a6            ; remove locals
        rts                     ; return

```

ENDFUNC

HaarXTopBwd FUNC EXPORT

```

        link       a6,#0          ; no local variables

        movea.l    $0008(a6),a0    ; start
        move.l     $000C(a6),d3    ; area
        lsr.l      #1,d3           ; area (long)
        subq.l     #1,d3           ; area-=1
        @do        move.l    (a0),d0 ; d0=HG=Y
        move.l     d0,d1          ; d1=HG
        swap       d1             ; d1=GH
        neg.w      d0             ; d0=H(-G)
        add.l      d1,d0          ; d0=01
        move.l     d0,(a0)+       ; *Y++=01
        dbf        d3,edo         ; while -1!--area

        unlk       a6            ; remove locals
        rts                     ; return

```

ENDFUNC

HaarTopBwd FUNC EXPORT

```

        link       a6,#0          ; no local variables
        movem.l    d4-d6,-(a7)    ; store registers

        movea.l    $0008(a6),a0    ; startH
        movea.l    a0,a1          ; startG
        move.l     $000C(a6),d6    ; height
        move.l     $0010(a6),d3    ; width
        move.l     d3,d6           ; linewidth=width
        add.l      d6,d6           ; linewidth (bytes)
        lsr.l      #1,d4          ; height/=2
        lsr.l      #1,d3          ; width/=2
        subq.l     #1,d4          ; height-=1
        subq.l     #1,d3          ; width-=1
        @do1       adda.l    d6,a1 ; startG+=linewidth
        move.l     d3,d5          ; linecount=width
        @do2       move.l    (a0),d0 ; d0=HAGB=Y0
        move.l     (a1),d1        ; d1=GAGB=Y1
        move.l     d0,d2          ; d2=HAGB
        add.l      d1,d0          ; d0=0A0B
        sub.l      d1,d2          ; d2=1A1B

        move.l     d0,d1          ; d1=HG
        swap       d1             ; d1=GH
        neg.w      d0             ; d0=H(-G)
        add.l      d1,d0          ; d0=01
        move.l     d0,(a0)+       ; *Y0++=0A0B

        move.l     d2,d1          ; d1=HG
        swap       d1             ; d1=GH

```

- 711 -

Engineering:KlicsCode:CompPict:Haar.a

```
neg.w    =d2                ; d2=W(-G)
add.l    d1,d2              ; d2=01
move.l    d2,(a1)+          ; *Y1++=1A1B

dbf       d5,@do2           ; while -1!--linecount
move.l    a1,a0              ; startH=startG
dbf       d4,@dol           ; while -1!--height

movem.l   (a7)+,d4-d6        ; restore registers
unlk      a6                 ; remove locals
rts                           ; return
```

ENDFUNC

END

- 712 -

Engineering:KlicsCode:CompPict:ConvolveSH3.c

```

.....
*
* Copyright 1993 KLICS Limited
* All rights reserved.
*
* Written by: Adrian Lewis
*
...../
**
2D wavelet transform convolver (fast hardware emulation)
New improved wavelet coeffs : 11 19 5 3

Optimized for speed:
    dirn = False
    src/dst octave == 0
*/

#define FwdS(addr0,dAG,dAH) \
    v=(short *)addr0; \
    dAG=(v3=v+(vs=v<<1)); \
    dAG+=v+(vs<<=1); \
    dAH=v3+(vs<<=1); \
    dAH+=v3+(vs<<=1);

#define Fwd1(addr1,dAG,dAH,dBG,dBH) \
    v=(short *)addr1; \
    dBG=(v3=v+(vs=v<<1)); \
    dAH+=v+(vs<<=1); \
    dBH=v3+(vs<<=1); \
    dAG+=v3+(vs<<=1);

#define Fwd2(addr2,addr1,addr0,dAG,dAH,dBG,dBH) \
    v=(short *)addr2; \
    dAH=(v3=v+(vs=v<<1)); \
    dBG+=v+(vs<<=1); \
    dAG+=v3+(vs<<=1); \
    dBH+=v3+(vs<<=1); \
    *(short *)addr0=(dAH+15)>>5; \
    *(short *)addr1=(dAG+15)>>5;

#define Fwd3(addr3,dAG,dAH,dBG,dBH) \
    v=(short *)addr3; \
    dAG=(v3=v+(vs=v<<1)); \
    dBH+=v+(vs<<=1); \
    dAH=v3+(vs<<=1); \
    dBG+=v3+(vs<<=1);

#define Fwd0(addr0,addr3,addr2,dAG,dAH,dBG,dBH) \
    v=(short *)addr0; \
    dBH=(v3=v+(vs=v<<1)); \
    dAG+=v+(vs<<=1); \
    dBG+=v3+(vs<<=1); \
    dAH+=v3+(vs<<=1); \
    *(short *)addr2=(dBH+15)>>5; \
    *(short *)addr3=(dBG+15)>>5;

#define FwdE(addr3,addr2,dBG,dBH) \
    v=(short *)addr3; \
    dBH=(vs=v<<1); \
    dBG=(vs<<2); \
    *(short *)addr2=(dBH+15)>>5; \
    *(short *)addr3=(dBG+15)>>5;

```

Engineering:KlicsCode:CompFict:ConvolveSH3.c

```

#define Fwd(base, end, inc) \
    addr0=base; \
    addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addr1=addr2-(inc>>2); \
    FwdS(addr0,dAG,dAH); \
    addr1+=inc; \
    Fwd1(addr1,dAG,dAH,dBG,dBH); \
    addr2+=inc; \
    Fwd2(addr2,addr1,addr0,dAG,dAH,dBG,dBH); \
    addr3+=inc; \
    while(addr3<end) { \
        Fwd3(addr3,dAG,dAH,dBG,dBH); \
        addr0+=inc; \
        Fwd0(addr0,addr3,addr2,dAG,dAH,dBG,dBH); \
        addr1+=inc; \
        Fwd1(addr1,dAG,dAH,dBG,dBH); \
        addr2+=inc; \
        Fwd2(addr2,addr1,addr0,dAG,dAH,dBG,dBH); \
        addr3+=inc; \
    } \
    FwdE(addr3,addr2,dBG,dBH);

extern void FASTFORWARD(char *data, long incl, long end1, long inc2, char *end2);
extern void HAARFORWARD(char *data, long incl, long end1, long inc2, char *end2);

void FastForward(char *data, long incl, long end1, long inc2, char *end2)
{
    register short v, vs, v3, dAG, dAH, dBG, dBH, inc;
    register char *addr0, *addr1, *addr2, *addr3, *end;
    char *base;

    inc=incl;
    for(base=data;base<end2;base+=inc2) {
        end=base+end1;
        Fwd(base,end,inc);
    }
}

void Daub4Forward(short *data, int size[2], int oct_dst)
{
    int oct, area=size[0]*size[1]<<1;
    short width=size[0]<<1;
    char *top=area+(char *)data, *left=width+(char *)data;

    for(oct=0;oct!=oct_dst;oct++) {
        long cinc=2<<oct, cinc4=cinc<<2,
            rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t.

        FASTFORWARD((char *)data,cinc4,width-cinc,rinc,top);
        FASTFORWARD((char *)data,rinc4,area-rinc,cinc,left);
    }
}

void HaarForward(short *data, int size[2], int oct_dst)
{
    int oct, area=size[0]*size[1]<<1;
    short width=size[0]<<1;
    char *top=area+(char *)data, *left=width+(char *)data;

    for(oct=0;oct!=oct_dst;oct++) {
        long cinc=2<<oct, cinc2=cinc<<1.

```

- 714 -

Engineering:KilcsCode:CompPict:ConvoiveSH3.c

```

rinc=size[0]<<oct+1, rinc2=rinc<<1; /* col and row increments in t

HAARFORWARD((char *)data,cinc2,width,rinc,top);
HAARFORWARD((char *)data,rinc2,area,cinc,left);
}

void Hybr.dForward(short *data, int size[2], int oct_dst)
{
    int    oct, area=size[0]*size[1]<<1;
    short  width=size[0]<<1;
    char    *top=area-(char *)data, *left=width+(char *)data;

    HAARFORWARD((char *)data,4,width,size[0]<<1,top);
    HAARFORWARD((char *)data,size[0]<<2,area,2,left);
    for(oct=1;oct!=oct_dst;oct++) {
        long    cinc=2<<oct, cinc4=cinc<<2,
                rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t

        FASTFORWARD((char *)data,cinc4,width-cinc,rinc,top);
        FASTFORWARD((char *)data,rinc4,area-rinc,cinc,left);
    }

#define BwdS0(addr0,dAG,dAH,dBH) \
    v=(short *)addr0; \
    dAG= -(v3=v+(vs=v<<1)); \
    dAH=v+(vs<<1); \
    dBH=vs<<1; \

#define BwdS1(addr1,addr0,dAG,dAH,dBH) \
    v=(short *)addr1; \
    dBH+=(vs=v<<1); \
    v3=vs+v; \
    dAG+=v3-(vs<<2); \
    dAH-=v3-(vs<<1); \
    *(short *)addr0=(dBH+3)>>3;

#define Bwd2(addr2,dAG,dAH,dBG,dBH) \
    v=(short *)addr2; \
    dBG= -(v3=v+(vs=v<<1)); \
    dBH=v+(vs<<1); \
    dAH+=v3-(vs<<1); \
    dAG+=v3-(vs<<1);

#define Bwd3(addr3,addr2,addr1,dAG,dAH,dBG,dBH) \
    v=(short *)addr3; \
    dAH+=(v3=v+(vs=v<<1)); \
    dAG+=v+(vs<<1); \
    dBG+=v3-(vs<<1); \
    dBH+=v3-(vs<<1); \
    *(short *)addr1=(dAH+7)>>4; \
    *(short *)addr2=(dAG+7)>>4;

#define Bwd0(addr0,dAG,dAH,dBG,dBH) \
    v=(short *)addr0; \
    dAG= -(v3=v+(vs=v<<1)); \
    dAH=v+(vs<<1); \
    dBH+=v3-(vs<<1); \
    dBG+=v3-(vs<<1);

#define Bwd1(addr1,addr0,addr3,dAG,dAH,dBG,dBH) \
    v=(short *)addr1; \

```

- 715 -

Engineering:KlinsCode:CompPict:ConvolveSH3.C

```

    dBH+=(v3)+v+(vs<<=1)); \
    DBG+=v+(vs<<=1); \
    dAG+=v3+(vs<<=1); \
    dAH+=v3+(vs<<=1); \
    *(short *)addr3=(dBH+7)>>4; \
    *(short *)addr0=(DBG+7)>>4;

#define BwdE2(addr3,dAG,dAH,dBH) \
    v=(short *)addr2; \
    v3=v+(vs<<=1); \
    dBH=(vs<<=2); \
    dAH+=v3+vs; \
    dAG+=v3+(vs<<=1);

#define BwdE3(addr3,addr2,addr1,dAG,dAH,dBH) \
    v=(short *)addr3; \
    dAH+=(v3)+v+(vs<<=1)); \
    dAG+=v+(vs<<=1); \
    dBH+=v3+(vs<<=1); \
    dBH+=v3+(vs<<=1); \
    *(short *)addr1=(dAH+7)>>4; \
    *(short *)addr2=(dAG+7)>>4; \
    *(short *)addr3=(dBH+3)>>3;

#define Bwd(base,end,inc) \
    addr0=base; \
    addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addr1=addr2-(inc>>2); \
    BwdS0(addr0,dAG,dAH,dBH); \
    addr1+=inc; \
    BwdS1(addr1,addr0,dAG,dAH,dBH); \
    addr2+=inc; \
    while(addr2<end) { \
        Bwd2(addr2,dAG,dAH,dBG,dBH); \
        addr3+=inc; \
        Bwd3(addr3,addr2,addr1,dAG,dAH,dBG,dBH); \
        addr0+=inc; \
        Bwd0(addr0,dAG,dAH,dBG,dBH); \
        addr1+=inc; \
        Bwd1(addr1,addr0,addr3,dAG,dAH,dBG,dBH); \
        addr2+=inc; \
    } \
    BwdE2(addr2,dAG,dAH,dBH); \
    addr3+=inc; \
    BwdE3(addr3,addr2,addr1,dAG,dAH,dBH);

extern void FASTBACKWARD(char *data, long incl, long loop1, long inc2, char *end2)
extern void HAAREBACKWARD(char *data, long incl, long loop1, long inc2, long loop2)
extern void HAARTOPBWD(char *data, long height, long width);
/* extern void HAARXTOPBWD(char *data, long area); */

void FastBackward(char *data, long incl, long endl, long inc2, char *end2)
{
    register short v, vs, v3, dAG, dAH, DBG, dBH, inc;
    register char *addr0, *addr1, *addr2, *addr3, *end;
    char *base;

    inc=incl;
    for(base=data; base<endl; base+=inc2) {
        end=base+endl;
        Bwd(base, end, inc);
    }
}

```

- 716 -

Engineering:KlicsCode:CompPict:ConvolveSH3.c

```

)

void Daub4Backward(short *data, int size[2], int oct_src)
{
    int    oct, area=size[0]*size[1]<<1;
    short  width=size[0]<<1;
    char   *top=area+(char *)data, *left=width+(char *)data;

    for(oct=oct_src-1; oct>=0; oct--) {
        long    cinc=2<<oct, cinc4=cinc<<2,
                rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t

        FASTBACKWARD((char *)data, rinc4, area-(rinc<<1), cinc, left);
        FASTBACKWARD((char *)data, cinc4, width-(cinc<<1), rinc, top);
    }

void HaarBackward(data, size, oct_src)

short  *data;
int     size[2], oct_src;

{
    int    oct, area=size[0]*size[1]<<1;
    short  width=size[0]<<1;
    char   *top=area+(char *)data, *left=width+(char *)data;

    for(oct=oct_src-1; oct>=0; oct--) {
        long    cinc=2<<oct, cinc2=cinc<<1,
                rinc=size[0]<<oct+1, rinc2=rinc<<1; /* col and row increments in t

        HAARBACKWARD((char *)data, rinc2, size[1]>>oct, cinc, size[0]>>oct);
        HAARBACKWARD((char *)data, cinc2, size[0]>>oct, rinc, size[1]>>oct);
    }
    HAARTOPBWD((char *)data, size[1], size[0]);
    /* HAARXTOPBWD((char *)data, area>>1); */
}

void HybridBackward(data, size, oct_src)

short  *data;
int     size[2], oct_src;

{
    int    oct, area=size[0]*size[1]<<1;
    short  width=size[0]<<1;
    char   *top=area+(char *)data, *left=width+(char *)data;

    for(oct=oct_src-1; oct>=0; oct--) {
        long    cinc=2<<oct, cinc4=cinc<<2,
                rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t

        FASTBACKWARD((char *)data, rinc4, area-(rinc<<1), cinc, left);
        FASTBACKWARD((char *)data, cinc4, width-(cinc<<1), rinc, top);
    }
    HAARTOPBWD((char *)data, size[1], size[0]);
    /* HAARXTOPBWD((char *)data, area>>1); */
}

```

- 717 -

Engineering:KlicsCode:CompPict:ConvolveSH3.a

© Copyright 1993 KLICS Limited
All rights reserved.

Written by: Adrian Lewis

68000 FastForward/Backward code

seg 'klics'

macro
FwdStart &addr0,&dAG,&dAH

```

move.w    (&addr0),d0    ; v=(short *)addr0
move.w    d0,d1          ; vs=v
add.w     d1,d1          ; vs<<=1
move.w    d1,d2          ; vj=vs
add.w     d0,d2          ; vj=vs+v
move.w    d2,&dAG        ; dAG=vj
add.w     d1,d1          ; vs<<=1
add.w     d0,&dAG        ; dAG+=v
add.w     d1,&dAG        ; dAG+=vs
move.w    d2,&dAH        ; dAH=vj
add.w     d1,d1          ; vs<<=1
add.w     d1,&dAH        ; dAH+=vs
add.w     d2,&dAH        ; dAH+=vj
add.w     d1,d1          ; vs<<=1
add.w     d1,&dAH        ; dAH+=vs

```

endm

macro
FwdOdd &addr1,&dAG,&dAH,&DBG,&DBH

```

move.w    (&addr1),d0    ; v=(short *)addr1
move.w    d0,d1          ; vs=v
add.w     d1,d1          ; vs<<=1
move.w    d1,d2          ; vj=vs
add.w     d0,d2          ; vj=vs+v
move.w    d2,&DBG        ; DBG=vj
add.w     d1,d1          ; vs<<=1
add.w     d0,&dAH        ; dAH+=v
add.w     d1,&dAH        ; dAH+=vs
move.w    d2,&DBH        ; DBH=vj
add.w     d1,d1          ; vs<<=1
add.w     d1,&DBH        ; DBH+=vs
sub.w     d2,&dAG        ; dAG-=vj
add.w     d1,d1          ; vs<<=1
sub.w     d1,&dAG        ; dAG-=vs

```

endm

macro
FwdEven &addr2,&addr1,&addr0,&dAG,&dAH,&DBG,&DBH

```

move.w    (&addr2),d0    ; v=(short *)addr2
move.w    d0,d1          ; vs=v
add.w     d1,d1          ; vs<<=1
move.w    d1,d2          ; vj=vs

```

- 718 -

Engineering:KlicsCode:CompPict:ConvolveSH3.a

```

add.w    d0,d2      ; v3=vs+v
sub.w    d2,&dAH     ; dAH-=v3
add.w    d1,d1      ; vs<=1
add.w    d0,&DBG     ; DBG+=v
add.w    d1,&DBG     ; DBG+=vs
add.w    d2,&dAG     ; dAG+=v3
add.w    d1,d1      ; vs<=1
add.w    d1,&dAG     ; dAG+=vs
add.w    d2,&DBH     ; DBH+=v3
add.w    d1,d1      ; vs<=1
add.w    d1,&DBH     ; DBH+=vs
clr.w    d0         ; d0=0
asr.w    #5,&dAH     ; dAH>=5
addx.w   d0,&dAH     ; round dAH
asr.w    #5,&dAG     ; dAG>=5
addx.w   d0,&dAG     ; round dAG
move.w   &dAH,(&addr0) ; *(short *)addr0=dAH
move.w   &dAG,(&addr1) ; *(short *)addr1=dAG

```

mend

```

macro
FwdEnd    &addr3,&addr2,&DBG,&DBH

```

```

move.w    (&addr3),d0      ; v=*(short *)addr3
add.w     d0,d0            ; v<=1
add.w     d0,&DBH          ; DBH+=v
lsl.w     #2,d0            ; v<=2
sub.w     d0,&DBG          ; DBG-=v
clr.w     d0              ; d0=0
asr.w     #5,&DBH          ; DBH>=5
addx.w    d0,&DBH          ; round DBH
asr.w     #5,&DBG          ; DBG>=5
addx.w    d0,&DBG          ; round DBG
move.w    &DBH,(&addr2)    ; *(short *)addr2=DBH
move.w    &DBG,(&addr3)    ; *(short *)addr3=DBG

```

endm.

```

macro
Fwd      &base,&end,&inc

```

```

movea.l   &base,a0          ; addr0=base
move.l    &inc,d0           ; d0=inc
asr.l     #2,d0             ; d0=inc>>2
movea.l   a0,a3            ; addr3=addr0
suba.l    d0,a3             ; addr3-=inc>>2
movea.l   a3,a2            ; addr2=addr3
suba.l    d0,a2             ; addr2-=inc>>2
movea.l   a2,a1            ; addr1=addr2
suba.l    d0,a1             ; addr1-=inc>>2
FwdStart  a0,d4,d5         ; FwdStart(addr0,dAG,dAH)
adda.l    &inc,a1           ; addr1+=inc
FwdOdd    a1,d4,d5,d6,d7    ; FwdOdd(addr1,dAG,dAH,DBG,DBH)
adda.l    &inc,a2           ; addr2+=inc
FwdEven   a2,a1,a0,d4,d5,d6,d7 ; FwdEven(addr2,addr1,addr0,dAG,dAH,DBH)
adda.l    &inc,a3           ; addr3+=inc
FwdOdd    a3,d6,d7,d4,d5    ; FwdOdd(addr3,DBG,DBH,dAG,dAH)
adda.l    &inc,a0           ; addr0+=inc
FwdEven   a0,a3,a2,d6,d7,d4,d5 ; FwdEven(addr0,addr3,addr2,DBG,DBH,dAG)
adda.l    &inc,a1           ; addr1+=inc
FwdOdd    a1,d4,d5,d6,d7    ; FwdOdd(addr1,dAG,dAH,DBG,DBH)
adda.l    &inc,a2           ; addr2+=inc

```

- 719 -

Engineering:Kl:csCode:CompPict:ConvoiveSH3.a

```

FwdEven      a2,a0,d4,d5,d6,d7      ; FwdEven(addr2,addr1,addr0,dAG,dAH,dB
adda.l       &inc,a3                 ; addr3++inc
cmpa.l       a3,&end                 ; addr3<end
bgt.w        @do                     ; while
FwdEnd       a3,a2,d6,d7             ; FwdEnd(addr3,addr2,dBG,dBH)

endm
-----
FastForward FUNC      EXPORT
link          a6,#0                  ; no local variables
movem.l       d4-d7/a3-a5,-(a7)      ; store registers

move.l        $000C(a6),d3           ; inc=inc1
movea.l       $0008(a6),a5           ; base=data
@do           movea.l       a5,a4     ; end=base
adda.l        $0010(a6),a4           ; end++end1
Fwd           a5,a4,d3              ; Fwd(base,end,inc)
adda.l        $0014(a6),a5           ; base++inc2
cmpa.l        $0018(a6),a5           ; end2>base
blt.w         @do                   ; for

movem.l       (a7)+,d4-d7/a3-a5      ; restore registers
unlk          a6                    ; remove locals
rts           ; return

ENDFUNC
-----
macro
BwdStart0      &addr0,&dAG,&dAH,&dBH

move.w        (&addr0),d0           ; v=(short *)addr0
move.w        d0,d1                 ; vs=v
add.w         d1,d1                 ; vs<<=1 (vs=2v)
add.w         d1,d0                 ; v+=vs (v=3v)
move.w        d0,&dAG               ; dAG=v3
neg.w         &dAG                  ; dAG=-dAG
move.w        d0,&dAH               ; dAH=v
add.w         d1,&dAH               ; dAH+=vs
lsl.w         #2,d1                 ; vs<<=2 (vs=8v)
move.w        d1,&dBH               ; dBH=vs

endm
-----
macro
BwdStart1      &addr1,&addr0,&dAG,&dAH,&dBH

move.w        (&addr1),d0           ; v=(short *)addr1
move.w        d0,d1                 ; vs=v
add.w         d1,d1                 ; vs<<=1
add.w         d1,&dBH               ; dBH+=vs
add.w         d1,d0                 ; v+=vs (v=3v)
lsl.w         #2,d1                 ; vs<<=2 (vs=8v)
add.w         d1,d0                 ; v+=vs (v=11v)
add.w         d0,&dAG               ; dAG+=v
add.w         d1,d0                 ; v+=vs (v=19v)
sub.w         d0,&dAH               ; dAH-=v
clr.w         d0                    ; d0=0
asr.w         #3,&dBH               ; dBH>>=3
addx.w        d0,&dBH               ; round dBH
move.w        &dBH,(&addr0)         ; *(short *)addr0=dBH

endm

```


- 720 -

Engineering:KlicsCode:CompPict:ConvolveSH3.a

```

-----
macro
BwdEven &addr2,&dAG,&dAH,&DBG,&dBH

    move.w    (&addr2),d0      ; v=*(short *)&addr2
    move.w    d0,d1           ; vs=v
    add.w     d1,d1           ; vs<<=1 (vs=2v)
    add.w     d1,d0           ; v+=vs (v=3v)
    move.w    d0,&DBG         ; DBG=v
    neg.w     &DBG            ; DBG=-DBG
    move.w    d0,&dBH         ; dBH=v
    add.w     d1,&dBH         ; dBH+=vs
    lsl.w     #2,d1           ; vs<<=2 (vs=8v)
    add.w     d1,d0           ; v+=vs (v=11v)
    add.w     d0,&dAH         ; dAH+=v
    add.w     d1,d0           ; v+=vs (v=19v)
    add.w     d0,&dAG         ; dAG+=v

endm

-----
macro
BwdOdd &addr3,&addr2,&addr1,&dAG,&dAH,&DBG,&dBH

    move.w    (&addr3),d0     ; v=*(short *)&addr3
    move.w    d0,d1           ; vs=v
    add.w     d1,d1           ; vs<<=1 (vs=2v)
    add.w     d1,d0           ; v+=vs (v=3v)
    add.w     d0,&dAH         ; dAH+=v
    add.w     d0,&dAG         ; dAG+=v
    add.w     d1,&dAG         ; dAG+=vs
    lsl.w     #2,d1           ; vs<<=2 (vs=8v)
    add.w     d1,d0           ; v+=vs (v=11v)
    add.w     d0,&DBG         ; DBG+=v
    add.w     d1,d0           ; v+=vs (v=19v)
    sub.w     d0,&dBH         ; dBH-=v
    clr.w     d0              ; d0=0
    asr.w     #4,&dAH         ; dAH>>=4
    addx.w    d0,&dAH         ; round dAH
    move.w    &dAH,(&addr1)   ; *(short *)&addr1=dAH
    asr.w     #4,&dAG         ; dAG>>=4
    addx.w    d0,&dAG         ; round dAG
    move.w    &dAG,(&addr2)   ; *(short *)&addr2=dAG

endm

-----
macro
BwdEnd2 &addr2,&dAG,&dAH,&dBH

    move.w    (&addr2),d0     ; v=*(short *)&addr2
    move.w    d0,d1           ; vs=v
    add.w     d1,d1           ; vs<<=1 (vs=2v)
    add.w     d1,d0           ; v+=vs (v=3v)
    lsl.w     #2,d1           ; vs<<=2 (vs=8v)
    move.w    d1,&dBH         ; dBH=vs
    add.w     d1,d0           ; v+=vs (v=11v)
    add.w     d0,&dAH         ; dAH+=v
    add.w     d1,d0           ; v+=vs (v=19v)
    add.w     d0,&dAG         ; dAG+=v

endm

-----
macro
BwdEnd? &addr3,&addr2,&addr1,&dAG,&dAH,&DBG

```

- 721 -

Engineering: K1icsCode: CompPict: ConvolveSH3.a

```

move.w    (&addr3),d0      ; v=(short *)addr3
move.w    d0,d1            ; vs=v
add.w     d1,d1            ; vs<<=1 (vs=2v)
add.w     d1,d0            ; v+=vs (v=3v)
add.w     d0,&dAH          ; dAH+=v
add.w     d0,&dAG          ; dAG+=v
add.w     d1,&dAG          ; dAG+=vs
add.w     d1,&dBH          ; dBH+=vs
lsl.l     #4,d1            ; vs<<=4 (v=32v)
sub.w     d1,&dBH          ; dBH-=vs
clr.w     d0               ; d0=0
asr.w     #4,&dAH          ; dAH>>=4
addx.w    d0,&dAH          ; round dAH
move.w    &dAH,(&addr1)    ; *(short *)addr1=dAH
asr.w     #4,&dAG          ; dAG>>=4
addx.w    d0,&dAG          ; round dAG
move.w    &dAG,(&addr2)    ; *(short *)addr2=dAG
asr.w     #3,&dBH          ; dBH>>=3
addx.w    d0,&dBH          ; round dBH
move.w    &dBH,(&addr3)    ; *(short *)addr3=dBH

endm

-----
macro
Bwd      &base,&end,&inc
.
movea.l  &base,a0          ; addr0=base
move.l   &inc,d0           ; d0=inc
asr.l    #2,d0             ; d0=inc>>2
movea.l  a0,a3             ; addr3=addr0
suba.l   d0,a3             ; addr3-=inc>>2
movea.l  a3,a2             ; addr2=addr3
suba.l   d0,a2             ; addr2-=inc>>2
movea.l  a2,a1             ; addr1=addr2
suba.l   d0,a1             ; addr1-=inc>>2
BwdStart0 a0,d4,d5,d7      ; BwdStart0(addr0,dAG,dAH,dBH)
adda.l   &inc,a1           ; addr1+=inc
BwdStart1 a1,a0,d4,d5,d7   ; BwdStart1(addr1,addr0,dAG,dAH,dBH)
adda.l   &inc,a2           ; addr2+=inc
@do
BwdEven  a2,d4,d5,d6,d7    ; BwdEven(addr2,dAG,dAH,dBG,dBH)
adda.l   &inc,a3           ; addr3+=inc
BwdOdd   a3,a2,a1,d4,d5,d6,d7 ; BwdOdd(addr3,addr2,addr1,dAG,dAH,dBG)
adda.l   &inc,a0           ; addr0+=inc
BwdEven  a0,d6,d7,d4,d5    ; BwdEven(addr0,dBG,dBH,dAG,dAH)
adda.l   &inc,a1           ; addr1+=inc
BwdOdd   a1,a0,a3,d6,d7,d4,d5 ; BwdOdd(addr1,addr0,addr3,dBG,dBH,dAG)
adda.l   &inc,a2           ; addr2+=inc
crrpa.l  a2,&end           ; addr2<end
bgt      @do              ; while
BwdEnd2  a2,d4,d5,d7       ; BwdEnd2(addr2,dAG,dAH,dBH)
adda.l   &inc,a3           ; addr3+=inc
BwdEnd3  a3,a2,a1,d4,d5,d7 ; BwdEnd3(addr3,addr2,addr1,dAG,dAH,dB)

endm
-----
FastBackward  FUNC      EXPORT
.
link          a6,#0       ; no local variables
movem.l       d4-d7/a3-a5,-(a7) ; store registers

move.l        S000C(a6),d3 ; inc=inc1
movea.l       S0008(a6),a5 ; base=data

```

- 722 -

Engineering:KlicsCode:CompPact:ConvolveSH3.a

```

0do      movea.l    a5,a4          : end=base
         adda.l     $0010(a6),a4    : end+=end1
         Bwd        a5,a4,d3        : Bwd(base,end,inc)
         adda.l     $0014(a6),a5     : base+=inc2
         cmpa.l     $0018(a6),a5     : end2>base
         blt.w      0do             : for

         movem.l    (a7)+,d4-d7/a3-a5 : restore registers
         unlink     a6              : remove locals
         rts                    : return

         ENDFUNC
-----
         END

```

- 723 -

Engineering:KlicsCode:CompPict:Colour.c

```

.....
*
*   © Copyright 1993 KLICS Limited
*   All rights reserved.
*
*   Written by: Adrian Lewis
*
*...../
/*
*   Test versions of colour space conversions in C
*/

#include <Memory.h>
#include <QuickDraw.h>

#define NewPointer(ptr,type,size) \
    saveZone=GetZone(); \
    SetZone(SystemZone()); \
    if (nil==(ptr=(type)NewPtr(size))) { \
        SetZone(ApplicZone()); \
        if (nil==(ptr=(type)NewPtr(size))) { \
            SetZone(saveZone); \
            return(MemoryError()); \
        } \
    } \
    SetZone(saveZone);

typedef union {
    long    pixel;
    char    rgb[4];
} Pixel;

/* Special YUV space version */
#define rgb_yuv(pixmap,Yc) \
    pixel.pixel=0x808080**pixmap++; \
    r=(short)pixel.rgb[1]; \
    g=(short)pixel.rgb[2]; g+=g; \
    b=(short)pixel.rgb[3]; \
    Y=(b<3)-b; \
    g+=r; \
    Y+=g+g-g; \
    Y>>=4; \
    Y=g; \
    *Yc++=Y; \
    Y>>=2; \
    U=b-Y; \
    V=r-Y;

#define limit(Y,low,high) \
    Y<(low<<2)?low<<2:Y>(high<<2)?high<<2:Y

/* Standard YUV space version - Bt294 CR07(0) mode limiting */
#define rgb_yuv32(pixmap,Yc) \
    pixel.pixel=0x808080**pixmap++; \
    r=(long)pixel.rgb[1]; \
    g=(long)pixel.rgb[2]; \
    b=(long)pixel.rgb[3]; \
    Y= (306*r + 601*g + 117*b)>>8; \
    *Yc++ = limit(Y,16-128,235-128); \
    U= (512*r - 429*g - 83*b)>>8; \
    V= (-173*r + 339*g + 512*b)>>8;

void RGB2YUV32(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int wid

```

Engineering:KlacsCode:CompPict:Colour.c

```

long    *pixmap2=pixmap+cols, *row, *end=pixmap+area;
short    *Yc2=Yc+width;

while(pixmap<end) {
    row=pixmap+width;
    while(pixmap<row) {
        Pixel pixel;
        long    r,g,b,Y,U=0,V=0;

        rgb_yuv32(pixmap,Yc);
        rgb_yuv32(pixmap,Yc);
        rgb_yuv32(pixmap2,Yc2);
        rgb_yuv32(pixmap2,Yc2);
        U>>=2;
        V>>=2;
        *Uc++=limit(U,16-128,240-128);
        *Vc++=limit(V,16-128,240-128);
    }
    pixmap+=cols+cols-width;
    pixmap2+=cols+cols-width;
    Yc+=width;
    Yc2+=width;
}

typedef struct {
    short    ry, rv, by, bu;
} RGB_Tab;

OSErr RGBTable(long **tab)
{
    RGB_Tab *table;
    int    i;
    THz    saveZone;

    NewPointer(table,RGB_Tab*,256*sizeof(RGB_Tab));
    *tab=(long *)table;
    for(i=0;i<128;i++) {
        table[i].ry=306*i>>8;
        table[i].rv=173*i>>8;
        table[i].by=117*i>>8;
        table[i].bu=83*i>>8;
    }
    for(i=128;i<256;i++) {
        table[i].ry=306*(i-256)>>8;
        table[i].rv=173*(i-256)>>8;
        table[i].by=117*(i-256)>>8;
        table[i].bu=83*(i-256)>>8;
    }
    return(noErr);
}

typedef struct {
    short    ru, gu, bv, gv;
} UV32_Tab;

UV32_Tab *UV32_Table()
{
    UV32_Tab    *table;
    int    i;

    table=(UV32_Tab *)NewPtr(256*sizeof(UV32_Tab));

```

Engineering:KillsCode:CompPict:Colour.c

```

for(i=0;i<128;i++) {
    table[i].ru=128+(1436*i>>10);
    table[i].gu=128+(-731*i>>10);
    table[i].bv=128+(1815*i>>10);
    table[i].gv=-352*i>>10;
}
for(i=128;i<256;i++) {
    table[i].ru=128-(1436*(i-256)>>10);
    table[i].gu=128+(-731*(i-256)>>10);
    table[i].bv=128+(1815*(i-256)>>10);
    table[i].gv=-352*(i-256)>>10;
}
return(table);
}

typedef struct {
    long    u, v;
} UV32Tab;

OSErr  UV32Table(long **tab)
{
    long    *ytab;
    UV32Tab *uvtab;
    int     i;
    THz     saveZone;

    NewPointer(*tab, long*, 512*sizeof(long)+512*sizeof(UV32Tab));
    ytab=*tab;
    uvtab=(UV32Tab*)&ytab[512];
    for(i=-256;i<256;i++) {
        long    yyy, sp;

        sp=0x000000fe&(i<-128?0:i>127?255:i+128);
        yyy=sp; yyy<=&8;
        yyy|=sp; yyy<=&8;
        yyy|=sp;
        ytab[0x000001ff&i]=yyy;
    }
    for(i=-256;i<256;i++) {
        long    ru,gu,bv,gv;

        ru=0xffffffff&(1436*i>>10);
        gu=0x000001fe&(-731*i>>10);
        bv=0x000001fe&(1815*i>>10);
        gv=0x000001fe&(-352*i>>10);

        uvtab[0x000001ff&i].u=((ru<&8)|gu)<&8;
        uvtab[0x000001ff&i].v=((gv<&8)|bv);
    }
    return(noErr);
}

typedef struct {
    short    u, v;
} UV16Tab;

OSErr  UV16Table(long **tab)
{
    short    *ytab;
    UV16Tab *uvtab;
    int     i;
    THz     saveZone;

```

- 726 -

Engineering:KlicsCode:CompPict:Colour.c

```

NewPointer(*tab, long*, 512*sizeof(short)-512*sizeof(UV16Tab));
ytab=(short **)tab;
uvtab=(UV16Tab*)&ytab[512];
for(i=-256; i<256; i++) {
    long    yyy, sp;

    sp=0x0000001e&((i<-128?0:i>127?255:i-128)>>3);
    yyy=sp; yyy<=5;
    yyy|=sp; yyy<=5;
    yyy|=sp;
    ytab[0x000001ff&i]=yyy;

    for(ii=-256; ii<256; ii++) {
        long    ru, gu, bv, gv;

        ru=0xffffffff & (1436*i>>13);
        gu=0x00000003e & (-731*i>>13);
        bv=0x00000003e & (1815*i>>13);
        gv=0x00000003e & (-352*i>>13);

        uvtab[0x000001ff&i].u=((ru<<5)|gu)<<5;
        uvtab[0x000001ff&i].v=(gv<<5)|bv;
    }
    return(noErr);
}

#define over(val) \
    ((0xFF00&(val)) == 0)?(char)val:val<0?0:255

/* Standard YUV space version */
#define yuv_rgb32(pixmap, Yc) \
    Y=(*Yc++)>>2; \
    pixel.rgb[1]=over(Y+r); \
    pixel.rgb[2]=over(Y-g); \
    pixel.rgb[3]=over(Y+b); \
    *pixmap++=pixel.pixel;

void    YUV2RGB32(long *pixmap, short *Yc, short *Uc, short *Vc, int area, int wid)
{
    long    *pixmap2=pixmap+cols.*row.*end=pixmap+area;
    short    *Yc2=Yc+width;

    while(pixmap<end) {
        row=pixmap+width;
        while(pixmap<row) {
            Pixel    pixel;
            long    r,g,b,Y,U,V;

            U=(*Uc++)>>2;
            V=(*Vc++)>>2;
            r=128+(1436*U>>10);
            g=128+(-731*U - 352*V>>10);
            b=128+(1815*V>>10);

            yuv_rgb32(pixmap, Yc);
            yuv_rgb32(pixmap, Yc);
            yuv_rgb32(pixmap2, Yc2);
            yuv_rgb32(pixmap2, Yc2);
        }
        pixmap+=cols+cols-width;
        pixmap2+=cols+cols-width;
        Yc+=width;
    }
}

```

- 727 -

Engineering:KlicsCode:CompPict:Colour.c

```

    Yc2+=width;
}

#define rgb32_yuv(pixmap,Yc) \
    pixel.pixel=0x808080~*pixmap++; \
    r=pixel.rgb[1]; \
    g=pixel.rgb[2]; \
    b=pixel.rgb[3]; \
    Y= (table[0xFF&r].ry + (g<<2)-table[0xFF&g].ry-table[0xFF&g].by + table[0xFF&b] \
    *Yc++ = limit(Y,16-128,235-128); \
    U+= (r<<1) -g -table[0xFF&g].rv - table[0xFF&b].bu; \
    V+= (b<<1) -g -table[0xFF&r].rv - table[0xFF&g].bu;

void RGB32YUV(RGB_Tab *table,long *pixmap, short *Yc, short *Uc, short *Vc, int
{
    long *pixmap2=pixmap+cols, *row, *end=pixmap+area;
    short *Yc2=Yc+width;

    while(pixmap<end) {
        row=pixmap+width;
        while(pixmap<row) {
            Pixel pixel;
            long r,g,b,Y,U=0,V=0;

            rgb32_yuv(pixmap,Yc);
            pixel.pixel=0x808080~*pixmap++;
            r=pixel.rgb[1];
            g=pixel.rgb[2];
            b=pixel.rgb[3];
            Y= (table[0xFF&r].ry + (g<<2)-table[0xFF&g].ry-table[0xFF&g].by + tabl
            *Yc++ = limit(Y,16-128,235-128);
            U+= (r<<1) -g -table[0xFF&g].rv - table[0xFF&b].bu;
            V+= (b<<1) -g -table[0xFF&r].rv - table[0xFF&g].bu;

            rgb32_yuv(pixmap,Yc);
            rgb32_yuv(pixmap2,Yc2);
            rgb32_yuv(pixmap2,Yc2);
            U>>=2;
            V>>=2;
            *Uc++=limit(U,16-128,240-128);
            *Vc++=limit(V,16-128,240-128);
        }
        pixmap+=cols+cols-width;
        pixmap2+=cols+cols-width;
        Yc+=width;
        Yc2+=width;
    }

#define yuv_rgb32x2(pixmap,Y) \
    pixel.rgb[1]=over(Y+r); \
    pixel.rgb[2]=over(Y+g); \
    pixel.rgb[3]=over(Y+b); \
    pixmap[cols]=pixel.pixel; \
    *pixmap++=pixel.pixel;

void YUV2RGB32x2(UV32_Tab *table,long *pixmap, short *Yc, short *Uc, short *Vc,
{
    long *pixmap2=pixmap+2*cols, *row, *end=pixmap+area;
    short *Yc2=Yc+width;

```


Engineering:KlicsCode:CompPict:Colour.c

```

while(pixmap<end) {
    long    Yold=*Yc>>2, Yold2=*Yc2>>2;

    row=pixmap+width*2;
    while(pixmap<row) {
        Pixel pixel;
        long    r,g,b,Y,U,V;

        U=0x00FF&((*Uc++)>>2);
        V=0x00FF&((*Vc++)>>2);
        r=table[U].ru;
        g=table[U].gu+table[V].gv;
        b=table[V].bv;

        Y=(*Yc++)>>2;
        Yold=(Y+Yold)>>1;
        yuv_rgb32x2(pixmap,Yold);

        Yold=Y;
        yuv_rgb32x2(pixmap,Yold);

        Y=(*Yc++)>>2;
        Yold=(Y+Yold)>>1;
        yuv_rgb32x2(pixmap,Yold);

        Yold=Y;
        yuv_rgb32x2(pixmap,Yold);

        Y=(*Yc2++)>>2;
        Yold2=(Y+Yold2)>>1;
        yuv_rgb32x2(pixmap2,Yold2);

        Yold2=Y;
        yuv_rgb32x2(pixmap2,Yold2);

        Y=(*Yc2++)>>2;
        Yold2=(Y+Yold2)>>1;
        yuv_rgb32x2(pixmap2,Yold2);

        Yold2=Y;
        yuv_rgb32x2(pixmap2,Yold2);
    }
    pixmap+=4*cols-2*width;
    pixmap2+=4*cols-2*width;
    Yc+=width;
    Yc2+=width;
}

#define yuv_rgb8(pixel,Yc,index,dith) \
    Y=*Yc++; \
    Y<<=3; \
    Y&= 0x3F00; \
    Y|= U; \
    pixel.rgb[index]=table[Y].rgb[dith];

void YUV2RGB8(Pixel *table,long *pixmap, short *Yc, short *Uc, short *Vc, int a
(
    long    *pixmap2=pixmap+cols*1, *row, *end=pixmap+area/4;
    short    *Yc2=Yc+width;

    while(pixmap<end) {

```

- 729 -

Engineering:KlicsCode:CcmpPict:Colour.c

```

row=pixmap*width/4;
while(pixmap<row) {
    Pixel pixel, pixel2;
    long Y,U,V;

    U=*Uc++;
    V=*Vc++;
    U>>=2;
    V>>=6;
    U= (U&0xF0) | (V&0x0F);

    yuv_rgb8(pixel,Yc,0,3);
    yuv_rgb8(pixel,Yc,1,0);
    yuv_rgb8(pixel2,Yc2,0,1);
    yuv_rgb8(pixel2,Yc2,1,2);

    U=*Uc++;
    V=*Vc++;
    U>>=2;
    V>>=6;
    U= (U&0xF0) | (V&0x0F);

    yuv_rgb8(pixel,Yc,2,3);
    yuv_rgb8(pixel,Yc,3,0);
    yuv_rgb8(pixel2,Yc2,2,1);
    yuv_rgb8(pixel2,Yc2,3,2);

    *pixmap++=pixel.pixel;
    *pixmap2++=pixel2.pixel;
}
pixmap+=(cols+cols-width)/4;
pixmap2+=(cols+cols-width)/4;
Yc+=width;
Yc2+=width;
}

#define yuv_rgb8x2(pixel,pixel2,Y,index,dith,dith2) \
    Y!= 0x3F00; \
    Y!= 0; \
    pixel.rgb[index]=table[Y].rgb[dith]; \
    pixel2.rgb[index]=table[Y].rgb[dith2];

void YUV2RGB8x2(Pixel *table, long *pixmap, short *Yc, short *Uc, short *Vc, int
(
    long *pixmap2=pixmap+cols/2, *row, *end=pixmap+area/4;
    short *Yc2=Yc+width;

    while(pixmap<end) {
        long Yold=*Yc<<3, Yold2=*Yc2<<3;

        row=pixmap*width/2;
        while(pixmap<row) {
            Pixel pixel, pixel2, pixel3, pixel4;
            long Y,U,V;

            U=*Uc++;
            V=*Vc++;
            U>>=2;
            V>>=6;
            U= (U&0x00F0) | (V&0x000F);

            Y=(*Yc++)<<3;

```

- 730 -

Engineering:KlacsCode:CompPict:Colour.c

```

Yold=(Y+Yold)>>1;
yuv_rgb8x2(pixel,pixel2,Y,0,3,1);

Yold=Y;
yuv_rgb8x2(pixel,pixel2,Y,1,0,2);
Yold=Y;

Y=(*Yc++)<<3;
Yold=(Y+Yold)>>1;
yuv_rgb8x2(pixel,pixel2,Y,2,3,1);

Yold=Y;
yuv_rgb8x2(pixel,pixel2,Y,3,0,2);
Yold=Y;

Y=(*Yc2++)<<3;
Yold2=(Y+Yold2)>>1;
yuv_rgb8x2(pixel3,pixel4,Y,0,3,1);

Yold2=Y;
yuv_rgb8x2(pixel3,pixel4,Y,1,0,2);
Yold2=Y;

Y=(*Yc2++)<<3;
Yold2=(Y+Yold2)>>1;
yuv_rgb8x2(pixel3,pixel4,Y,2,3,1);

Yold2=Y;
yuv_rgb8x2(pixel3,pixel4,Y,3,0,2);
Yold2=Y;

pixmap[cols/4]=pixel2.pixel;
*pixmap++=pixel.pixel;

pixmap2[cols/4]=pixel4.pixel;
*pixmap2++=pixel3.pixel;
}
pixmap+=(cols+cols-width)/2;
pixmap2+=(cols+cols-width)/2;
Yc+=width;
Yc2+=width;
}

#define yuv_rgbTEST(pixel,index,Y) \
    rgb_col.red=(Y+r<<8); \
    rgb_col.green=(Y+g<<8); \
    rgb_col.blue=(Y+b<<8); \
    pixel.rgb[index]=Color2Index(&rgb_col);

void YUV2RGBTEST(UV32_Tab *table,long *pixmap, short *Yc, short *Uc, short *Vc,
{
    long *pixmap2=pixmap+cols/2, *row, *end=pixmap+area/4;
    short *Yc2=Yc+width;

    while(pixmap<end) {
        long Yold=*Yc<<3, Yold2=*Yc2<<3;

        row=pixmap+width/2;
        while(pixmap<row) {
            RGBColor rgb_col;
            Pixel pixel, pixel2;

```

- 731 -

Engineering:KlicsCode:CompPict:Colour.c

```

long r,g,b,Y,U,V;

U=0x00FF&((*Uc++)>>2);
V=0x00FF&((*Vc++)>>2);
r=table[U].ru;
g=table[U].gu+table[V].gv;
b=table[V].bv;

Y=(*Yc++)>>2;
Yold=(Y+Yold)>>1;
rgb_col.red=(Yold+r<<8);
rgb_col.green=(Yold+g<<8);
rgb_col.blue=(Yold+b<<8);
pixel.rgb[0]=Color2Index(&rgb_col);

Yold=Y;
yuv_rgbTEST(pixel,1,Yold);

Y=(*Yc++)>>2;
Yold=(Y+Yold)>>1;
yuv_rgbTEST(pixel,2,Yold);

Yold=Y;
yuv_rgbTEST(pixel,3,Yold);

Y=(*Yc2++)>>2;
Yold2=(Y+Yold2)>>1;
yuv_rgbTEST(pixel2,0,Yold2);

Yold2=Y;
yuv_rgbTEST(pixel2,1,Yold2);

Y=(*Yc2++)>>2;
Yold2=(Y+Yold2)>>1;
yuv_rgbTEST(pixel2,2,Yold2);

Yold2=Y;
yuv_rgbTEST(pixel2,3,Yold2);

pixmap[cols/4]=pixel.pixel;
*pixmap++=pixel.pixel;

pixmap2[cols/4]=pixel2.pixel;
*pixmap2++=pixel2.pixel;
}
pixmap+=(cols+cols-width)/2;
pixmap2+=(cols+cols-width)/2;
Yc+=width;
Yc2+=width;

```

- 732 -

Engineering:KlicsCode:CompPict:Colour.a

```

-----
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
-----
*
*  68030 Colour space conversions
*
-----
machine mc68030
seg      'klics'
include  'Traps.a'
-----
macro
DPY32x2      &LARGB, &row, &o0, &ol, &n0, &n1

    add.l      &n0, &o0
    lsr.l      #1, &o0
    add.l      &n1, &ol
    lsr.l      #1, &ol
    ; interpolate first pixel

    move.l     &o0, (&LARGB)
    add.l      &row, &LARGB
    move.l     &o0, (&LARGB)
    add.l      &row, &LARGB
    move.l     &ol, (&LARGB)
    add.l      &row, &LARGB
    move.l     &ol, (&LARGB)+

    move.l     &n1, (&LARGB)
    sub.l      &row, &LARGB
    move.l     &n1, (&LARGB)
    sub.l      &row, &LARGB
    move.l     &n0, (&LARGB)
    sub.l      &row, &LARGB
    move.l     &n0, (&LARGB)+

endm

macro
DPY32      &LARGB, &row, &o0, &ol, &n0, &n1

    move.l     &o0, (&LARGB)
    add.l      &row, &LARGB
    move.l     &ol, (&LARGB)+

    move.l     &n1, (&LARGB)
    sub.l      &row, &LARGB
    move.l     &n0, (&LARGB)+

endm

-----
macro
UV2RGB32      &AU, &AV, &TAB

    add.l      #2048, &TAB
    ; move to uvtab

    move.w     &AU, d1
    lsr.w      #2, d1
    and.w      #501FF, d1
    ; Load U

```

- 733 -

Engineering:KlicsCode:CompPict:Colour.a

```

move.l    (&TAB,d1.w*8),d0      ; UV now rg (u)

move.w    &AV,d1                ; Load V
lsr.w     #2,d1
and.w     #01FF,d1
add.l     4(&TAB,d1.w*8),d0      ; UV now rgb

move.l     d0,d1                ; 3 copies
move.l     d0,d2
move.l     d0,d3

sub.l     #2048,&TAB             ; restore ytab

endm
-----
macro
GETY32      &AY, &TAB, &RGB0, &RGB1

move.l     &AY,d4                ; Y
lsr.w     #2,d4
and.w     #01FF,d4
add.l     (&TAB,d4.w*4),&RGB1    ; RGB1+=YYY

swap      d4
lsr.w     #2,d4
and.w     #01FF,d4
add.l     (&TAB,d4.w*4),&RGB0    ; RGB0+=YYY

endm
-----
macro
OVER32      &RGB

move.l     &RGB,d4                ; copy pixel
andi.l     #01010100,d4          ; was it this rgb
beq.s      @nx_rgb                ; if not then quit
btst       #24,d4                 ; R overflow?
beq.s      @bit16                ; if not then continue
btst       #23,&RGB               ; test sign
beq.s      @pos23                ; if positive
andi.l     #0000ff,&RGB           ; underflow sets R to 0
@bit16     bra.s                  ; do next bit
@pos23     ori.l                  ; overflow sets R to 255
@bit16     btst                   ; G overflow?
beq.s      @bit8                 ; if not then continue
btst       #15,&RGB               ; test sign
beq.s      @pos16                ; if positive
andi.w     #00ff,&RGB             ; underflow sets G to 0
bra.s      @bit8                 ; do next bit
@pos16     ori.w                  ; overflow sets G to 255
@bit8      btst                   ; B overflow?
beq.s      @end                  ; if not then continue
btst       #7,&RGB               ; test sign
seq        &RGB                  ; under/over flow
@end       andi.l                 ; mask RGB ok
@nx_rgb

endm
-----
macro
HASHOUT32   &AH, &D0, &D1, &D2, &D3

move.l     &D0,d4

```

- 734 -

Engineering:KlicsCode:CompPict:Colour.a

```

add.l    >    &D1,d4
add.l    &D2,d4
add.l    &D3,d4
andi.l   *$03e3e3e0,d4
move.l   d4,&AH

endm
-----
macro
HASHCMP32    &AH, &D0, &D1, &D2, &D3

move.l    &D0,d4
add.l    &D1,d4
add.l    &D2,d4
add.l    &D3,d4
andi.l   *$03e3e3e0,d4
cmp.l    &AH,d4

endm
-----
OUT32X2 FUNC      EXPORT
*
PS        RECORD      8
table     DS.L         1
pixmap    DS.L         1
Y         DS.L         1
U         DS.L         1
V         DS.L         1
width     DS.L         1
height    DS.L         1
rowByte   DS.L         1
pixmap2   DS.L         1
ENDR
*
LS        RECORD      0,DECR
Yl        DS.L         1      ; sizeof(short)*Yrow      = 2*width
U_ex      DS.L         1      ; x end address        = U-U_ix
U_ey      DS.L         1      ; y end address        = U*width+height>>
U_ix      DS.L         1      ; sizeof(short)*UVrow  = width
Y_y       DS.L         1      ; sizeof(short)*Yrow   = 2*width
P_y       DS.L         1      ; 4*rowBytes-sizeof(long)*Prow = 4*rowBytes-width
LSize     EQU          *
ENDR
*
a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7

link      a6,*LS.LSize      ; inc, width, fend and rowend are loca
movem.l   d4-d7/a3-a5,-(a7) ; store registers

move      SR,d0

move.l    PS.Y(a6),a0      ; Y=Yc
move.l    PS.U(a6),a1      ; U=Uc
move.l    PS.V(a6),a2      ; V=Vc
move.l    PS.pixmap(a6),a3 ; pm=pixmap
move.l    PS.table(a6),a4  ; tab=table
move.l    PS.pixmap2(a6),a5 ; pm2=pixmap2

move.l    PS.width(a6),d0   ; LOAD width
move.l    d0,LS.U_ix(a6)    ; SAVE U_ix
move.l    PS.height(a6),d1  ; LOAD height
mulu.w    d0,d1            ; width*height

```

- 735 -

Engineering:KlicsCode:CompPict:Colour.a

```

lsr.l    #1,d1          ; width*height/2
add.l    #1,d1          ; U+width*height/2
move.l    d1,LS_U_ey(a6) ; SAVE U_ey
add.l    d0,d0          ; width*2
move.l    d0,LS_Y1(a6)   ; SAVE Y1
move.l    d0,LS_Y_y(a6)  ; SAVE Y_y
lsr.l    #2,d0          ; width*8
move.l    PS.rowByte(a6),d1 ; LOAD rowBytes
lsr.l    #2,d1          ; rowBytes*4
sub.l    d0,d1          ; rowBytes*4-width*8
move.l    d1,LS_P_y(a6)  ; SAVE P_y

move.l    PS.rowByte(a6),d5 ; load rowBytes
clr.l    d6              ; clear old2
clr.l    d7              ; clear old1

@do_y    move.l    LS_U_ix(a6),d0 ; LOAD U_ixB
add.l    a1,d0           ; P+U_ixB
move.l    d0,LS_U_ex(a6) ; SAVE U_exB

@do_x    UV2RGB32    (a1)+,(a2)+,a4 ; uv2rgb(*U++,*V++)

move.l    LS_Y1(a6),d4 ; load Yrow
GETY32    (a0,d4.1),a4,d2,d3 ; add Yb to RGB values
GETY32    (a0)+,a4,d0,d1 ; add Ya to RGB values

move.l    d0,d4
or.l     d1,d4
or.l     d2,d4
or.l     d3,d4
andi.l    #01010100,d4
bne.s    @over          ; if overflow

@ok      HASHOUT32    (a5)+,d0,d1,d2,d3

DPY32x2    a3,d5,d6,d7,d0,d2
DPY32x2    a3,d5,d0,d2,d1,d3

move.l    d1,d6 ; copy olds
move.l    d3,d7

cmpa.l    LS_U_ex(a6),a1
blt.w     @do_x

add.l     LS_Y_y(a6),a0
add.l     LS_P_y(a6),a3

cmpa.l    LS_U_ey(a6),a1
blt.w     @do_y

movem.l    (a7)+,d4-d7/a3-a5 ; restore registers
unlk      a6 ; remove locals
rts       ; return

@over    OVER32    d0
OVER32    d1
OVER32    d2
OVER32    d3
bra       @ok

```

ENDFUNC

```

-----
OUT32X2D  FUNC  EXPORT

```


- 736 -

Engineering:KlacsCode:CompPict:Colour.a

```

PS      RECORD      = 8
table   DS.L         1
pixmap  DS.L         1
Y        DS.L         1
U        DS.L         1
V        DS.L         1
width   DS.L         1
height  DS.L         1
rowByte DS.L         1
pixmap2 DS.L         1
        ENDR

LS      RECORD      0,DECR
Y1      DS.L         1      ; sizeof(short)*Yrow      = 2*width
U_ex    DS.L         1      ; x end address      = U+U_ix
U_ey    DS.L         1      ; y end address      = U*width+height>>
U_ix    DS.L         1      ; sizeof(short)*Uvrow = width
Y_y     DS.L         1      ; sizeof(short)*Yrow  = 2*width
P_y     DS.L         1      ; 4*rowBytes-sizeof(long)*Prow = 4*rowBytes-width
LSize   EQU
        ENDR

.
.      a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
.      d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
.
        link      a6,#LS.LSize      ; inc. width, fend and rowend are loca
        movem.l   d4-d7/p3-a5,-(a7) ; store registers

.
        move.l    PS.Y(a6),a0      ; Y=Yc
        move.l    PS.U(a6),a1      ; U=Uc
        move.l    PS.V(a6),a2      ; V=Vc
        move.l    PS.pixmap(a6),a3 ; pm=pixmap
        move.l    PS.table(a6),a4   ; tab=table
        move.l    PS.pixmap2(a6),a5 ; pm2=pixmap2

        move.l    PS.width(a6),d0   ; LOAD width
        move.l    d0,LS.U_ix(a6)    ; SAVE U_ix
        move.l    PS.height(a6),d1  ; LOAD height
        mulu.w    d0,d1             ; width*height
        lsr.l     #1,d1             ; width*height/2
        add.l     a1,d1             ; U*width*height/2
        move.l    d1,LS.U_ey(a6)    ; SAVE U_ey
        add.l     d0,d0             ; width*2
        move.l    d0,LS.Y1(a6)      ; SAVE Y1
        move.l    d0,LS.Y_y(a6)     ; SAVE Y_y
        lsl.l     #2,d0             ; width*8
        move.l    PS.rowByte(a6),d1 ; LOAD rowBytes
        lsl.l     #2,d1             ; rowBytes*4
        sub.l     d0,d1             ; rowBytes*4-width*8
        move.l    d1,LS.P_y(a6)     ; SAVE P_y

        move.l    PS.rowByte(a6),d5 ; load rowBytes
        clr.l     d6                ; clear old2
        clr.l     d7                ; clear old1

@do_y   move.l    LS.U_ix(a6),d0     ; LOAD U_ixB
        add.l     a1,d0             ; P+U_ixB
        move.l    d0,LS.U_ex(a6)    ; SAVE U_exB

@do_x   UV2RGB32   (a1)+,(a2)+,a4   ; uv2rgb(*U++,*V++)

        move.l    LS.Y1(a6),d4      ; load Yrow
        GETY32    (a0,d4.1),a4,d2,d3 ; add Yb to RGB values

```

- 737 -

Engineering:KlicsCode:CompPict:Colour.a

```

GETY32      (a0)+,a4,d0,d1      ; add YA to RGB values
move.l      d0,d4
or.l        d1,d4
or.l        d2,d4
or.l        d3,d4
andi.l      #01010100,d4
bne.w       @over              ; if overflow

@ok          HASHCMP32 (a5)+,d0,d1,d2,d3
bne.s       @diff

add.l       #16,a3              ; add four pixels

@cont        move.l      d1,d6      ; copy olds
move.l      d3,d7

cmpa.l      LS.U_ex(a6),a1
blt.w       @do_x

add.l       LS.Y_y(a6),a0
add.l       LS.P_y(a6),a3

cmpa.l      LS.U_ey(a6),a1
blt.w       @do_y

movem.l     (a7)+,d4-d7/a3-a5      ; restore registers
unlk        a6                    ; remove locals
rts          ; return

@diff        move.l      d4,-4(a5)
DPY32x2     a3,d5,d6,d7,d0,d2
DPY32x2     a3,d5,d0,d2,d1,d3
bra.s       @cont

@over        OVER32      d0
OVER32      d1
OVER32      d2
OVER32      d3
bra         @ok

```

ENDFUNC

OUT32	FUNC	EXPORT
PS	RECORD	8
table	DS.L	1
pixmap	DS.L	1
Y	DS.L	1
U	DS.L	1
V	DS.L	1
width	DS.L	1
height	DS.L	1
rowByte	DS.L	1
pixmap2	DS.L	1
	ENDR	

LS	RECORD	0,DECR		
Y1	DS.L	1	; sizeof(short)*Yrow	= 2*width
U_ex	DS.L	1	; x end address	= U+U_ix
U_ey	DS.L	1	; y end address	= U+width*height>>
U_ix	DS.L	1	; sizeof(short)*U/row	= width
Y_y	DS.L	1	; sizeof(short)*Yrow	= 2*width
P_y	DS.L	1	; 2*rowBytes-sizeof(long)*Prow	= 2*rowBytes-width
LSize	EQU	.		

- 738 -

Engineering:KlincsCode:CompPict:Colour.a

```

ENDR
.
.
a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - cld0, d7
.

link      a6, #LS.LSize      ; inc, width, fend and rowend are loca
movem.l   d4-d7/a3-a5, -(a7) ; store registers

move.l    PS.Y(a6), a0      ; Y=Yc
move.l    PS.U(a6), a1      ; U=Uc
move.l    PS.V(a6), a2      ; V=Vc
mcve.l    PS.pixmap(a6), a3 ; pm=pixmap
move.l    PS.table(a6), a4   ; tab=table
move.l    PS.pixmap2(a6), a5 ; pm2=pixmap2

move.l    PS.width(a6), d0   ; LOAD width
move.l    d0, LS.U_ix(a6)    ; SAVE U_ix
move.l    PS.height(a6), d1  ; LOAD height
mulu.w    d0, d1             ; width*height
lsr.l     #1, d1             ; width*height/2
add.l     a1, d1             ; U*width*height/2
move.l    d1, LS.U_ey(a6)    ; SAVE U_ey
add.l     d0, d0             ; width*2
move.l    d0, LS.Y1(a6)      ; SAVE Y1
move.l    d0, LS.Y_y(a6)     ; SAVE Y_y
add.l     d0, d0             ; width*4
move.l    PS.rowByte(a6), d1 ; LOAD rowBytes
add.l     d1, d1             ; rowBytes*2
sub.l     d0, d1             ; rowBytes*2-width*4
move.l    d1, LS.P_y(a6)     ; SAVE P_y

move.l    PS.rowByte(a6), d5 ; load rowBytes
move.l    LS.Y1(a6), d6      ; load Yrow

@do_y     move.l    LS.U_ix(a6), d7 ; LOAD U_ix8
add.l     a1, d7             ; P+U_ix8

@dc_x     UV2RGB32    (a1)+, (a2)+, a4 ; uv2rgb(*U++, *V++)

GETY32    (a0, d6, 1), a4, d2, d3 ; add Yb to RGB values
GETY32    (a0)+, a4, d0, d1 ; add Ya to RGB values

move.l    d0, d4
or.l      d1, d4
or.l      d2, d4
or.l      d3, d4
andi.l    #01010100, d4
bne.s     @over ; if overflow

@ok       HASHOUT32    (a5)+, d0, d1, d2, d3
DPY32     a3, d5, d0, d2, d1, d3

cmpa.l    d7, a1
blt.w     @do_x

add.l     LS.Y_y(a6), a0
add.l     LS.P_y(a6), a3

cmpa.l    LS.U_ey(a6), a1
blt.w     @do_y

movem.l    (a7)+, d4-d7/a3-a5 ; restore registers

```

- 739 -

Engineering:Kl:csCode:CompPict:Colour.a

```

        unlk      a6                ; remove locals
        rts                      ; return
@over   OVER32    d0
        OVER32    d1
        OVER32    d2
        OVER32    d3
        bra       @ok
        .
        .
        .
        ENDFUNC
        -----
OUT32D  FUNC      EXPORT
        .
PS      RECORD    8
table   DS.L      1
pixmap  DS.L      1
Y       DS.L      1
U       DS.L      1
V       DS.L      1
width   DS.L      1
height  DS.L      1
rowByte DS.L      1
pixmap2 DS.L      1
        ENDR
        .
LS      RECORD    0,DECR
Y1      DS.L      1
U_ex    DS.L      1
U_ey    DS.L      1
U_ix    DS.L      1
Y_y     DS.L      1
P_y     DS.L      1
LSize   EQU       *
        ENDR
        .
        a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
        d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - Y-row, d7
        .
        link      a6,@LS.LSize      ; inc, width, fend and rowend are loca
        movem.l   d4-d7/a3-a5,-(a7) ; store registers ..
        .
        move.l    PS.Y(a6),a0        ; Y=Yc
        move.l    PS.U(a6),a1        ; U=Uc
        move.l    PS.V(a6),a2        ; V=Vc
        move.l    PS.pixmap(a6),a3   ; pm=pixmap
        move.l    PS.table(a6),a4    ; tab=table
        move.l    PS.pixmap2(a6),a5  ; pm2=pixmap2
        .
        move.l    PS.width(a6),d0     ; LOAD width
        move.l    d0,LS.U_ix(a6)      ; SAVE U_ix
        move.l    PS.height(a6),d1    ; LOAD height
        mulu.w    d0,d1               ; width*height
        lsr.l     #1,d1               ; width*height/2
        add.l     a1,d1               ; U*width*height/2
        move.l    d1,LS.U_ey(a6)      ; SAVE U_ey
        add.l     d0,d0               ; width*2
        move.l    d0,LS.Y1(a6)        ; SAVE Y1
        move.l    d0,LS.Y_y(a6)       ; SAVE Y_y
        add.l     d0,d0               ; width*4
        move.l    PS.rowByte(a6),d1   ; LOAD rowBytes
        add.l     d1,d1               ; rowBytes*2
        sub.l     d0,d1               ; rowBytes*2-width*4
        move.l    d1,LS.P_y(a6)       ; SAVE P_y

```

- 740 -

Engineering:KlicsCode:CompPict:Colour.a

```

        move.l    > PS.rowByte(a6),d5      ; load rowBytes
        move.l    LS.Y1(a6),d6             ; load Yrow

@do_y   move.l    LS.U_ix(a6),d7           ; LOAD U_ixB
        add.l     a1,d7                    ; P+U_ixB

@do_x   UV2RGB32   (a1)+,(a2)+,a4         ; uv2rgb(*U++,*V++)

        move.l    LS.Y1(a6),d4             ; load Yrow
        GETY32    (a0,d6,1),a4,d2,d3      ; add Yb to RGB values
        GETY32    (a0)+,a4,d0,d1         ; add Ya to RGB values

        move.l    d0,d4
        or.l      d1,d4
        or.l      d2,d4
        or.l      d3,d4
        andi.l    #01010100,d4
        bne.s     @over                   ; if overflow

@ok      HASHCMP32 (a5)+,d0,d1,d2,d3
        bne.s     @diff

        addq      #8,a3                   ; add four pixels

@cont:   cmpa.l    d7,a1
        blt.w     @do_x

        add.l     LS.Y_y(a6),a0
        add.l     LS.P_y(a6),a3

        cmpa.l    LS.U_ey(a6),a1
        blt.w     @do_y

        movem.l   (a7)+,d4-d7/a3-a5      ; restore registers
        unlk      a6                     ; remove locals
        rts                          ; return

@diff    move.l    d4,-4(a5)
        DPY32     a3,d5,d0,d2,d1,d3
        bra.s     @cont

@over    OVER32    d0
        OVER32    d1
        OVER32    d2
        OVER32    d3
        bra       @ok

        -----
        macro
        UVOV      &VAL, &OV

        move.w     &VAL,&OV
        add.w      #0200,&OV
        and.w      #5FC00,&OV
        beq.s      @ok
        tst.w      &OV
        bge.s      @pos
        move.w     #01FF,&VAL
        bra.s      @ok
        move.w     #5FE00,&VAL

@pos
@ok

        endm

```

- 741 -

Engineering:KlicsCode:CompPict:Colour.a

```

UVLIMIT FUNC      EXPORT
* fix d0, d4, spare d1, d2
  UVOV            d0, d1
  swap            d0
  UVOV            d0, d1
  swap            d0
  UVOV            d4, d1
  swap            d4
  UVOV            d4, d1
  swap            d4
  rts
ENDFUNC

macro
UVOVER            &U, &V

  move.l          #02000200, d1
  move.l          d1, d2
  add.l           &U, d1
  add.l           &V, d2
  or.l            d2, d1
  andi.l          #5FC00FC00, d1
  beq.s           @UVok
  bsr             UVLIMIT
@UVok

endm

macro
GETUV            &AU, &AV, &SP, &UV

  move.l          (&AU)+, &SP
  move.l          (&AV)+, &UV
  UVOVER          &SP, &UV
  lsr.l           #5, &UV
  andi.l          #03e003e0, &SP
  andi.l          #001F001F, &UV
  or.l            &SP, &UV          ; UV==000UV000UV
  swap            &UV

endm

macro
GETY            &AY, &IND, &UV, &R0, &R1

  move.l          &AY, &R1          ; (2+) Y=Y0Y1
  lsl.l           #5, &R1          ; (4) Y=Y0XXY1XX
  andi.l          #5FC00FC00, &R1
  or.w            &UV, &R1          ; (2) Y=Y1UV
  move.l          (&IND, &R1 .w*4), &R0 ; (2+) R0=0123 (Y1)
  swap            &R1              ; (4) Y=Y0XX
  or.w            &UV, &R1          ; (2) Y=Y0UV
  move.l          (&IND, &R1 .w*4), &R1 ; (2+) R1=0123 (Y0)

endm

macro
UV8            &AU, &AV, &SP, &UV

  move.l          (&AU)+, &SP
  move.l          (&AV)+, &UV
  UVOVER          &SP, &UV

```

- 742 -

Engineering:KlicsCode:CompPict:Colour.a

```

lsr.l    >    #2.&SP
lsr.l    #6.&UV
andl.l   #00F000F0.&SP
andl.l   #000F000F.&UV
or.l     &SP.&UV          ; UV==S00UV00UV
swap     SUV

endm

macro
Y2IND    &Y.&IND.&UV.&D0.&D1

move.l   &Y.&D0          ; d0=Y0Y1
lsr.l    #3.&D0          ; d0=Y0XXY1XX
move.b   &UV.&D0          ; d0=Y0XXY1UV
andl.w   #03FFF.&D0      ; d0=0YUV(1)
move.l   (&IND.&D0 .w*4).&D1 ; find clut entries
swap     &D0             ; d0=Y0XX
move.b   &UV.&D0          ; d0=Y0UV
andl.w   #03FFF.&D0      ; d0=0YUV(0)
move.l   (&IND.&D0 .w*4).&D0 ; find clut entries

endm

OUT8      FUNC      EXPORT
*
PS         RECORD    8
table      DS.L      1
pixmap     DS.L      1
Y          DS.L      1
U          DS.L      1
V          DS.L      1
width      DS.L      1
height     DS.L      1
rowByte    DS.L      1
pixmap2    DS.L      1
ENDR

*
LS         RECORD    0,DECH
Y1         DS.L      1          ; sizeof(short)*Yrow          = 2*width
U_ex       DS.L      1          ; x end address              = U+U_ix
U_ey       DS.L      1          ; y end address              = U*width+height>>
U_ix       DS.L      1          ; sizeof(short)*UVrow       = width
Y_y        DS.L      1          ; sizeof(short)*Yrow        = 2*width
P_y        DS.L      1          ; 2*rowBytes-sizeof(long)*Prow = 2*rowBytes-width
LSize      EQU
ENDR

*
a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7

*
link       a6,#LS.LSize          ; inc. width, fend and rowend are loca
movem.l    d4-d7/a3-a5,-(a7)     ; store registers

*
move.l     PS.Y(a6),a0           ; Y=Yc
move.l     PS.U(a6),a1           ; U=Uc
move.l     PS.V(a6),a2           ; V=Vc
move.l     PS.pixmap(a6),a3      ; pm=pixmap
move.l     PS.table(a6),a4       ; tab=table
adda.l     #00020000,a4          ; tab+=32768 (longs)
move.l     PS.pixmap2(a6),a5     ; pm2=pixmap2

move.l     PS.width(a6),d0       ; LOAD width

```

- 743 -

Engineering:KlicsCode:CcmpPict:Colour.a

```

move.l    d0,LS.U_ix(a6)      ; SAVE U_ix
move.l    PS.height(a6),d1    ; LOAD height
mulu.w    d0,d1               ; width*height
lsr.l     #1,d1               ; width*height/2
add.l     a1,d1               ; U*width*height/2
move.l    d1,LS.U_ey(a6)      ; SAVE U_ey
move.l    PS.rowByte(a6),d1    ; LOAD rowBytes
add.l     d1,d1               ; rowBytes*2
sub.l     d0,d1               ; rowBytes*2-width
move.l    d1,LS.P_y(a6)       ; SAVE P_y
add.l     d0,d0               ; width*2
move.l    d0,LS.Y1(a6)        ; SAVE Y1
move.l    d0,LS.Y_y(a6)       ; SAVE Y_y

move.l    PS.rowByte(a6),d5    ; load rowBytes
move.l    LS.Y1(a6),d6        ; load Yrow

@do_y     move.l    LS.U_ix(a6),d7    ; LOAD U_ixB
add.l     a1,d7               ; P+U_ixB

@do_x     GETUV    a1,a2,d0,d4

GETY      (a0,d6.w),a4,d4,d2,d3 ; d2=X0XX, d3=XX1X
GETY      (a0)+,a4,d4,d0,d1    ; d0=XXX0, d1=1XXX

move.w    d3,d2               ; d2=X01X
lsr.l     #8,d2               ; d2=01XX
move.w    d0,d1               ; d1=1XX0
swap      d1                  ; d1=X01X
lsr.l     #8,d1               ; d1=01XX

swap      d4                   ; next UV

GETY      (a0,d6.l),a4,d4,d0,d3 ; d0=X2XX, d3=XX3X
move.w    d3,d0               ; d0=X23X
lsr.l     #8,d0               ; d0=XX23
move.w    d0,d2               ; d2=0123-
GETY      (a0)+,a4,d4,d0,d3    ; d0=XXX2, d3=3XXX
move.w    d0,d3               ; d3=3XX2
swap      d3                  ; d3=X23X
lsr.l     #8,d3               ; d3=XX23
move.w    d3,d1               ; d1=C123

move.l    d2,(a3,d5)
move.l    d1,(a3)+

cmpa.l    d7,a1
blt.w     @do_x

add.l     LS.Y_y(a6),a0
add.l     LS.P_y(a6),a3

cmpa.l    LS.U_ey(a6),a1
blt.w     @do_y

movem.l   (a7)+,d4-d7/a3-a5    ; restore registers
unlk      a6                   ; remove locals
rts       ; return

-----
macro
Y8x2      SAY,&IND,&UV,&old

```


- 744 -

Engineering:KlicsCode:CompPict:Colour.a

```

move.l    &AY.d0                ; (2+) Y=Y0Y1
lsl.l     #3.d0                 ; (4) Y=Y0XXY1XX
swap      d0                    ; (4) Y=Y1XXY0XX
add.w     d0,&old                ; (2) old=old+Y0
lsr.w     #1,&old                ; (4) old=(old+Y0)/2
move.b    &UV,&old              ; (2) old=Y10UV
andi.w    #53FFF,&old           ; (4) old=0YUV(I0)
move.l    (&IND,&old.w*4),d1     ; (2+) d1=X1X3
move.w     d0,&old              ; (2) old=Y0
move.b     &UV,d0               ; (2) Y=Y0UV
andi.w    #53FFF,d0            ; (4) Y=0YUV(0)
move.l    (&IND,d0.w*4),d2     ; (2+) d2=0X2X
move.w     d1,d3               ; (2) exg.w d1,d2
move.w     d2,d1               ; (2) d1=X12X
move.w     d3,d2               ; (2) d2=0XX3
swap      d2                    ; (4) d2=X30X
lsl.l     #8,d1                ; (4) d1=12XX
lsl.l     #8,d2                ; (4) d2=30XX
swap      d0                    ; (4) Y=Y1XX
add.w     d0,&old                ; (2) old=old+Y1
lsr.w     #1,&old                ; (4) old=(old+Y1)/2
move.b     &UV,&old              ; (2) old=Y11UV
andi.w    #53FFF,&old           ; (4) old=0YUV(I1)
move.l    (&IND,&old.w*4),d3     ; (2+) d3=X1X3
move.w     d0,&old              ; (2) old=Y1
move.b     &UV,d0               ; (2) Y=Y0UV
andi.w    #53FFF,d0            ; (4) Y=0YUV(0)
move.l    (&IND,d0.w*4),d0     ; (2+) d0=0X2X
move.w     d0,d1               ; (2) exg.w d0,d3
move.w     d3,d0               ; (2) d0=0XX3
move.w     d1,d3               ; (2) d3=X12X
swap      d0                    ; (4) d0=X30X
lsr.l     #8,d0                ; (4) d0=XX30
lsr.l     #8,d3                ; (4) d3=X12X
move.w     d0,d2               ; (2) d2=3030 (YiY0YiY1) (1)
move.w     d3,d1               ; (2) d1=2121 (YiY0YiY1) (2)

```

endm

macro

Y8x2a

&AY,&IND,&UV

GETY

&AY,&IND,&UV,d1,d2

move.l

&AY,d2

; (2+) Y=Y0Y1

lsl.l

#3,d2

; (4) Y=Y0XXY1XX

move.b

&UV,d2

; (2) Y=Y1UV

andi.w

#53FFF,d2

; (4) Y=0YUV(Y1)

move.l

(&IND,d2.w*4),d1

; (2+) d1=0123 (Y1)

swap

d2

; (4) Y=Y0XX

move.b

&UV,d2

; (2) Y=Y0UV

andi.w

#53FFF,d2

; (4) Y=0YUV(Y0)

move.l

(&IND,d2.w*4),d2

; (2+) d2=0123 (Y0)

move.w

d1,d0

; (2) exg.w d2,d1

move.w

d2,d1

; (2) d1=0123 (Y1Y0)

move.w

d0,d2

; (2) d2=0123 (Y0Y1)

swap

d1

; (4) d1=2301 (Y0Y1)

endm

macro

Y8x2b

&AY,&IND,&UV

GETY

&AY,&IND,&UV,d1,d2

- 745 -

Engineering:KlicsCode:CompPict:Colour.a

```

*      move.l    => &AY,d2      ; (2+) Y=Y0Y1
*      lsl.l     #3,d2          ; (4) Y=Y0XXY1XX
*      move.b    &UV,d2        ; (2) Y=Y1UV
*      andi.w     #S3FFF,d2     ; (4) Y=0YUV(Y1)
*      move.l     (&IND,d2.w*4).d1 ; (2+) d1=0123 (Y1)
*      swap      d2            ; (4) Y=Y0XX
*      move.b     &UV,d2        ; (2) Y=Y0UV
*      andi.w     #S3FFF,d2     ; (4) Y=0YUV(Y0)
*      move.l     (&IND,d2.w*4).d2 ; (2+) d2=0123 (Y0)
*      ror.l      #8,d2         ; (6) d2=3012 (Y0)
*      ror.l      #8,d1         ; (6) d1=3012 (Y1)
*      move.w     d1,d0         ; (2) exg.w d2,d1
*      move.w     d2,d1         ; (2) d1=3012 (Y1Y0)
*      move.w     d0,d2         ; (2) d2=3012 (Y0Y1)
*      swap      d1            ; (4) d1=1230 (Y0Y1)
*      ror.w      #8,d1         ; (6) d1=1203 (Y0Y1)
*
*      endm

```

OUT8x2 FUNC EXPORT

```

*
PS      RECORD      8
table   DS.L        1
pixmap  DS.L        1
Y       DS.L        1
U       DS.L        1
V       DS.L        1
width   DS.L        1
height  DS.L        1
rowByte DS.L        1
pixmap2 DS.L        1
*      ENDR

LS      RECORD      0,DECR
Y1      DS.L        1      ; sizeof(short)*Yrow      = 2*width
U_ex    DS.L        1      ; x end address          = U+U_ix
U_ey    DS.L        1      ; y end address          = U+width*height>>
U_ix    DS.L        1      ; sizeof(short)*Uvrow    = width
Y_y     DS.L        1      ; sizeof(short)*Yrow     = 2*width
P_y     DS.L        1      ; 4*rowBytes-sizeof(long)*Frow = 4*rowBytes-width
LSize   EQU
*      ENDR

*
*      a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
*      d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d5 - old0, d7
*
link     a6,#LS.LSize      ; inc. width, fend and rowend are loca
movem.l  d4-d7/a3-a5,-(a7) ; store registers

*
move.l   PS.Y(a6),a0      ; Y=Yc
move.l   PS.U(a6),a1      ; U=Uc
move.l   PS.V(a6),a2      ; V=Vc
move.l   PS.pixmap(a6),a3 ; pm=pixmap
move.l   PS.table(a6),a4  ; tab=table
adda.l   #S00020000,a4    ; tab+=32768 (longs)
move.l   PS.pixmap2(a6),a5 ; pm2=pixmap2

*
move.l   PS.width(a6),d0   ; LOAD width
move.l   d0,LS.U_ix(a6)    ; SAVE U_ix
move.l   PS.height(a6),d1  ; LOAD height
mulu.w   d0,d1            ; width*height
lsr.l    #1,d1            ; width*height/2

```

- 746 -

Engineering:KlicsCode:CompPict:Colour.a

```

add.l    a1,d1                : U.width*height/2
move.l   d1,LS.U_ey(a6)       : SAVE U_ey
add.l    d0,d0                : width*2
move.l   d0,LS.Y1(a6)         : SAVE Y1
move.l   d0,LS.Y_y(a6)        : SAVE Y_y
move.l   PS.rowByte(a6),d1    : LCAD rowBytes
add.l    d1,d1                : rowBytes*2
add.l    d1,d1                : rowBytes*4
sub.l    d0,d1                : rowBytes*4-width*2
move.l   d1,LS.P_y(a6)        : SAVE P_y

move.l   PS.rowByte(a6),d5    : load rowBytes
clr.l    d6
clr.l    d7

3do_y    move.l   LS.U_ix(a6),d0 : LOAD U_ixB
add.l    a1,d0                : P+U_ixB
move.l   d0,LS.U_ex(a6)       : SAVE U_exB

8do_x    GETUV               a1,a2,d0,d4 : d4=00UV00UV (10)
Y8x2a    (a0),a4,d4;.d6       : calc d2,d1 pixels
move.l   d2,(a3)
add.l    d5,a3
move.l   d1,(a3)
add.l    d5,a3

move.l   LS.Y1(a6),d0         : load Yrow
Y8x2b    (a0,d0.w),a4,d4;.d7   : calc d2,d1 pixels
move.l   d2,(a3)
add.l    d5,a3
move.l   d1,(a3)+

swap     d4                  : next UV
addq.l   #4,a0               : next Ys

move.l   LS.Y1(a6),d0         : load Yrow
Y8x2b    (a0,d0.w),a4,d4;.d7   : calc d2,d1 pixels
move.l   d1,(a3)
sub.l    d5,a3
move.l   d2,(a3)
sub.l    d5,a3

Y8x2a    (a0)+,a4,d4;.d6
move.l   d1,(a3)
sub.l    d5,a3
move.l   d2,(a3)+

cmpa.l   LS.U_ex(a6),a1
blt.w    8do_x

add.l    LS.Y_y(a6),a0
add.l    LS.P_y(a6),a3

cmpa.l   LS.U_ey(a6),a1
blt.w    8do_y

movem.l   (a7)+,d4-d7/a3-a5    : restore registers
unlk     a6                   : remove locals
rts      : return

```

ENDFUNC

- 747 -

Engineering:KlicsCode:CompPict:Colour.a

```

macro
RGB2Y      =  ;RGB,&Y,&U,&V,&AY

move.l     ;RGB,d2          ; pixel:=pixmap
ecr1.l     ;$808080,d2      ; pixel:=0x808080
clr.w      d1               ; B=0
move.b     d2,d1            ; B=pixel[3]
move.l     4(a4,d1.w*8),d0   ; d0=by,bu
sub.w      d0,&U             ; U=bu
swap       d0               ; d0=bu.by
move.w     d0,&Y             ; Y=by
ext.w      d1               ; (short)B
add.w      d1,d1             ; B*=2
add.w      d1,&V             ; V+=B<<1
lsr.l      #8,d2             ; pixel>>=8
clr.w      d1               ; G=0
move.b     d2,d1            ; G=pixel[3]
move.l     (a4,d1.w*8),d0    ; d0=gry,gv
sub.w      d0,&U             ; U=gv
swap       d0               ; d0=gv.gry
sub.w      d0,&Y             ; Y=gv.gry
move.l     4(a4,d1.w*8),d0    ; d0=gby,gu
sub.w      d0,&V             ; V=gv
swap       d0               ; d0=gu.gby
sub.w      d0,&Y             ; Y=gby
ext.w      d1               ; (short)G
sub.w      d1,&U             ; U=g
sub.w      d1,&V             ; V=g
lsr.l      #2,d1             ; G<<=2
add.w      d1,&Y             ; Y+=G<<1
lsr.l      #8,d2             ; pixel>>=8
move.l     (a4,d2.w*8),d0    ; d0=ry,rv
sub.w      d0,&V             ; V=rv
swap       d0               ; d0=rv.ry
add.w      d0,&Y             ; Y=rv.ry
ext.w      d2               ; (short)R
add.w      d2,d2             ; R*=2
add.w      d2,&U             ; U+=R<<2
cmpl.w     #$FE40,&Y          ; Y>=-448
bge.s      @ok               ; if greater
move.w     #$FE40,&Y          ; Y=-448
bra.s      @end              ; save
@ok         cmpl.w     #$01C0,&Y          ; Y< 448
blt.s      @end              ; if less
move.w     #$01C0,&Y          ; Y= 448
@end        move.w     &Y,&AY          ; Save Y

endm

```

IN32	FUNC	EXPORT
PS	RECORD	8
table	DS.L	1
pixmap	DS.L	1
Y	DS.L	1
U	DS.L	1
V	DS.L	1
width	DS.L	1
height	DS.L	1
rowByte	DS.L	1
.	ENDR	
LS	RECORD	0,DECR

- 748 -

Engineering:KlicsCode:CompPict:Colour.3

```

Y1      DS.L      = 1      ; sizeof(short)*Yrow      = 2*width
U_ex    DS.L      1      ; x end address          = U*U_ix
U_ey    DS.L      1      ; y end address          = U*width*height>>
U_ix    DS.L      1      ; sizeof(short)*Uvrow      = width
Y_y     DS.L      1      ; sizeof(short)*Yrow      = 2*width
P_y     DS.L      1      ; 2*rowBytes-sizeof(long)*Prow = 2*rowBytes-width
LSize   EQU      *
ENDR

;
; a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
; d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7
;
link     a6,#LS.LSize      ; inc, width, fend and rowend are loca
movem.l  d4-d7/a3-a5,-(a7) ; store registers

move.l   PS.Y(a6),a0      ; Y=Yc
move.l   PS.U(a6),a1      ; U=Uc
move.l   PS.V(a6),a2      ; V=Vc
move.l   PS.pixmap(a6),a3 ; pm=pixmap
move.l   PS.table(a6),a4  ; tab=table

move.l   PS.width(a6),d0   ; LOAD width
move.l   d0,LS.U_ix(a6)    ; SAVE U_ix
move.l   PS.height(a6),d1  ; LOAD height
mulu.w   d0,d1             ; width*height
lsr.l    #1,d1             ; width*height/2
add.l    a1,d1             ; U*width*height/2
move.l   d1,LS.U_ey(a6)    ; SAVE U_ey
add.l    d0,d0             ; width*2
move.l   d0,LS.Y1(a6)      ; SAVE Y1
move.l   d0,LS.Y_y(a6)     ; SAVE Y_y
add.l    d0,d0             ; width*4
move.l   PS.rowByte(a6),d1 ; LOAD rowBytes
add.l    d1,d1             ; rowBytes*2
sub.l    d0,d1             ; rowBytes*2-width*4
move.l   d1,LS.P_y(a6)     ; SAVE P_y

move.l   PS.rowByte(a6),d7 ; load rowBytes
move.l   LS.Y1(a6),d6      ; load Y1

@do_y    move.l   LS.U_ix(a6),d0 ; LOAD U_ixB
add.l    a1,d0             ; P-U_ixB
move.l   d0,LS.U_ex(a6)    ; SAVE U_exB

@do_x    clr.w     d4      ; U=0
clr.w     d5      ; V=0

RGB2Y    (a3,d7.w),d3,d4,d5,(a0,d6.w); Convert pixel
RGB2Y    (a3)+,d3,d4,d5,(a0)+      ; Convert pixel
RGB2Y    (a3,d7.w),d3,d4,d5,(a0,d6.w); Convert pixel
RGB2Y    (a3)+,d3,d4,d5,(a0)+      ; Convert pixel

asr.w     #2,d4      ; U>>=2
asr.w     #2,d5      ; V>>=2

cmpi.w    #5FE40,d4   ; U>=-448
bge.s     @okU        ; if greater
move.w    #5FE40,d4   ; U=-448
bra.s     @doV        ; save
@okU      cmpi.w    #501C0,d4 ; U< 448
blt.s     @doV        ; if less
move.w    #501C0,d4   ; U= 448

```

- 749 -

Engineering:KlicsCode:CompPict:Colour.a

```

2doV    cmpil.w    =>    #SFE40,d5            ; V>=-448
        bge.s      9okV          ; if greater
        move.w     #SFE40,d5        ; V= -448
        bra.s      @end          ; save
9okV     cmpil.w    #S01CC,d5        ; V< 448
        blt.s      @end          ; if less
        move.w     #S01C0,d5       ; V= 448

@end     move.w     d4,(a1)+        ; Save U
        move.w     d5,(a2)+        ; Save V

        cmpa.l     LS.U_ex(a6),a1
        blt.w      @do_x

        add.l      LS.Y_y(a6),a0
        add.l      LS.P_y(a6),a3

        cmpa.l     LS.U_ey(a6),a1
        blt.w      @do_y

        movem.l    (a7)+,d4-d7/a3-a5 ; restore registers
        unlk      a6              ; remove locals
        rts        ; return

```

ENDFUNC

```

-----
macro
UV16      &A0, &AV, &SP, &UV

        move.l     (&A0)+,&SP
        move.l     (&AV)+,&UV
        UOVER      &SP,&UV
        lsr.l      #5,&UV
        andi.l     #03e003e0,&SP
        andi.l     #001F001F,&UV
        or.l       &SP,&UV          ; UV==000UV00UV
        swap       &UV

        endm

macro
Y16x2     &AY,&IND,&UV

        move.l     &AY,d2          ; (2+) Y=Y0Y1
        lsl.l      #5,d2           ; (4) Y=Y0XXY1XX
        andi.l     #SFC00FC00,d2
        or.w       &UV,d2          ; (2) Y=Y1UV
        move.l     (&IND,d2.w*4),d1 ; (2+) d1=0123 (Y1)
        swap       d2             ; (4) Y=Y0XX
        or.w       &UV,d2          ; (2) Y=Y0UV
        move.l     (&IND,d2.w*4),d2 ; (2+) d2=0123 (Y0)

        endm

```

OUT16x2	FUNC	EXPORT
PS	RECORD	8
table	DS.L	1
pixmap	DS.L	1
Y	DS.L	1
U	DS.L	1
V	DS.L	1

- 750 -

Engineering:KlacsCode:CompPict:Colour.a

```

width DS.L      P
height DS.L     1
rowByte DS.L    1
pixmap2 DS.L    1
        ENDR

LS      RECORD      0,DECR
Y1      DS.L        1      ; sizeof(short)*Yrow      = 2*width
U_ex    DS.L        1      ; x end address      = U+U_ix
U_ey    DS.L        1      ; y end address      = U+width*height>>
U_ix    DS.L        1      ; sizeof(short)*UVrow  = width
Y_y     DS.L        1      ; sizeof(short)*Yrow  = 2*width
P_y     DS.L        1      ; 4*rowBytes-sizeof(long)*Prow = 4*rowBytes-width
LSize   EQU        *
        ENDR

        .
        a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
        d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7

        link      a6,#LS.LSize      ; inc. width, fend and rowend are loca
        movem.l   d4-d7/a3-a5,-(a7) ; store registers

        .
        move.l    PS.Y(a6),a0      ; Y=Yc
        move.l    PS.U(a6),a1      ; U=Uc
        move.l    PS.V(a6),a2      ; V=Vc
        move.l    PS.pixmap(a6),a3 ; pm=pixmap
        move.l    PS.table(a6),a4   ; tab=table
        adda.l    #500020000,a4     ; tab+=32768 (longs)
        move.l    PS.pixmap2(a6),a5 ; pm2=pixmap2

        .
        move.l    PS.width(a6),d0   ; LOAD width
        move.l    d0,LS.U_ix(a6)    ; SAVE U_ix
        move.l    PS.height(a6),d1  ; LOAD height
        mulu.w    d0,d1              ; width*height
        lsr.l     #1,d1              ; width*height/2
        add.l     a1,d1              ; U+width*height/2
        move.l    d1,LS.U_ey(a6)    ; SAVE U_ey
        add.l     d0,d0              ; width*2
        move.l    d0,LS.Y1(a6)      ; SAVE Y1
        move.l    d0,LS.Y_y(a6)     ; SAVE Y_y
        add.l     d0,d0              ; width*4
        move.l    PS.rowByte(a6),d1 ; LOAD rowBytes
        add.l     d1,d1              ; rowBytes*2
        add.l     d1,d1              ; rowBytes*4
        sub.l     d0,d1              ; rowBytes*4-width*4
        move.l    d1,LS.P_y(a6)     ; SAVE P_y

        .
        move.l    PS.rowByte(a6),d5 ; load rowBytes
        clr.l     d6
        clr.l     d7

@do_y    move.l    LS.U_ix(a6),d0     ; LOAD U_ixB
        add.l     a1,d0              ; P+U_ixB
        move.l    d0,LS.U_ex(a6)     ; SAVE U_exB

@do_x    GETOV     a1,a2,d0,d4        ; d4=00UV00UV (10)

        GETY      (a0),a4,d4,d1,d2   ; calc d2,d1 pixel
        move.l    d2,(a3)+
        move.l    d1,(a3)
        add.l     d5,a3
        swap      d1
        move.l    d1,(a3)

```

- 751 -

Engineering:KlicsCode:CcmpPict:Colour.a

```

swap      d2
move.l    d2, -(a3)
add.l     d5, a3

move.l    LS.Y1(a6), d0      ; load Yrow
GETY      (a0, d0.w), a4, d4, d1, d2 ; calc d2, d1 pixels
move.l    d2, (a3)+
move.l    d1, (a3)
add.l     d5, a3
swap      d1
move.l    d1, (a3)
swap      d2
move.l    d2, -(a3)

swap      d4                  ; next UV
addq.l    #4, a0              ; next Ys
add.l     #12, a3

move.l    LS.Y1(a6), d0      ; load Yrow
GETY      (a0, d0.w), a4, d4, d1, d2 ; calc d2, d1 pixels
move.l    d1, (a3)
move.l    d2, -(a3)
sub.l     d5, a3
swap      d2
move.l    d2, (a3)+
swap      d1
move.l    d1, (a3)
sub.l     d5, a3

GETY      (a0)+, a4, d4, d1, d2
move.l    d1, (a3)
move.l    d2, -(a3)
swap      d2
sub.l     d5, a3
move.l    d2, (a3)+
swap      d1
move.l    d1, (a3)+

cmpa.l    LS.U_ex(a6), a1
blt.w     @do_x

add.l     LS.Y_y(a6), a0
add.l     LS.P_y(a6), a3

cmpa.l    LS.U_ey(a6), a1
blt.w     @do_y

movem.l    (a7)+, d4-d7/a3-a5 ; restore registers
unlk      a6                  ; remove locals
rts        ; return

```

ENDFUNC

```

-----
macro
Y16      &AY, &IND, &UV

move.l    &AY, d2              ; (2+) Y=Y0Y1
lsl.l     #5, d2              ; (4) Y=Y0XXY1XX
andi.l    #5FC00FC00, d2      ;
or.w      &UV, d2              ; (2) Y=Y1UV
move.l    (&IND, d2.w*4), d1   ; (2+) d1=Y1
swap      d2                  ; (4) Y=Y0XX
or.w      &UV, d2              ; (2) Y=Y0UV

```


- 752 -

Engineering:KlicsCode:CompPict:Colour.a

```

move.l    (%IND.d2.w*4),d2      ; (2-) d2=Y0
move.w    d1,d2                ; (2) d2=Y0Y1

endm

CUT16     FUNC      EXPORT
.
PS        RECORD          8
table     DS.L           1
pixmap    DS.L           1
Y         DS.L           1
U         DS.L           1
V         DS.L           1
width     DS.L           1
height    DS.L           1
rowByte   DS.L           1
pixmap2   DS.L           1
.        ENDR

LS        RECORD          0,DECR
Y1        DS.L           1      ; sizeof(short)*Yrow      = 2*width
U_ex      DS.L           1      ; x end address          = U+U_ix
U_ey      DS.L           1      ; y end address          = U+width*height>>
U_ix      DS.L           1      ; sizeof(short)*UVrow    = width
Y_y       DS.L           1      ; sizeof(short)*Yrow     = 2*width
P_y       DS.L           1      ; 2*rowBytes-sizeof(long)*Prow = 2*rowBytes-width
LSize     EQU            *
.        ENDR

.
a0 - Y, a1 - U, a2 - V, a3 - pixmap, a4 - table, a5 - pixmap2
d0 - rgb00, d1 - rgb01, d2 - rgb10, d3 - rgb11, d4 - spare, d6 - old0, d7

link      a6,*LS.LSize      ; inc, width, fend and rowend are loca
movem.l   d4-d7/a3-a5,-(a7) ; store registers

move.l    PS.Y(a6),a0        ; Y=Yc
move.l    PS.U(a6),a1        ; U=Uc
move.l    PS.V(a6),a2        ; V=Vc
move.l    PS.pixmap(a6),a3   ; pm=pixmap
move.l    PS.table(a6),a4    ; tab=table
adda.l    #500020000,a4      ; tab->32768 (longs)
move.l    PS.pixmap2(a6),a5  ; pm2=pixmap2

move.l    PS.width(a6),d0     ; LOAD width
move.l    d0,LS.U_ix(a6)      ; SAVE U_ix
move.l    PS.height(a6),d1    ; LOAD height
mulu.w    d0,d1              ; width*height
lsr.l     #1,d1              ; width*height/2
add.l     a1,d1              ; U+width*height/2
move.l    d1,LS.U_ey(a6)      ; SAVE U_ey
add.l     d0,d0              ; width*2
move.l    d0,LS.Y1(a6)        ; SAVE Y1
move.l    d0,LS.Y_y(a6)       ; SAVE Y_y
move.l    PS.rowByte(a6),d1   ; LOAD rowBytes
add.l     d1,d1              ; rowBytes*2
sub.l     d0,d1              ; rowBytes*2-width*2
move.l    d1,LS.P_y(a6)       ; SAVE P_y

move.l    PS.rowByte(a6),d5    ; load rowBytes
clr.l     d6
clr.l     d7

edo_y     move.l    LS.U_ix(a6),d0      ; LOAD U_ixB

```


- 754 -

Engineering:KlicsCode:CompPict:Color2.a

```

*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.

```

```

*  Written by: Adrian Lewis
*

```

```

*  68000 Fast RGB/YUV code
*

```

```

include 'Traps.a'
machine mc68030

```

```

macro
RGB2Y    &Apixel,&AY

```

```

    d0 - pixel/r, d1 - g/2g+r, d2 - b, d3 - Y

```

```

    move.l    &Apixel,d0      ; pixel=&Apixel
    eor.l     #500808080,d0    ; signed pixels

```

```

    move.b    d0,d2           ; b=pixel[3]
    ext.w     d2              ; b is 8(16) bit

```

```

    move.w    d0,d1           ; g=pixel[2]
    asr.w     #7,d1           ; 2g is 9(16) bit

```

```

    swap      d0              ; r=pixel[1]
    ext.w     d0              ; r is 8(16) bit

```

```

    move.w    d2,d3           ; Y=b
    lsl.w     #3,d3           ; Y<<=3
    sub.w     d2,d3           ; Y-=b

```

```

    add.w     d0,d1           ; 2g+r
    add.w     d1,d3           ; Y+=2g+r
    add.w     d1,d3           ; Y+=2g+r
    add.w     d1,d3           ; Y+=2g+r
    asr.w     #4,d3           ; Y>>=4
    add.w     d1,d3           ; Y+=2g+r
    move.w    d3,&AY          ; AY=Y is 10(16) bit

```

```

    endm

```

```

macro
RGB2UV    &AU,&AV

```

```

    d0 - r, d2 - b, d3 - Y, d1 - U/V

```

```

    add.w     d0,d0           ; r is 9(16) bit
    add.w     d2,d2           ; b is 9(16) bit
    asr.w     #1,d3           ; Y is 9(16) bit
    move.w    d2,d1           ; U=b
    sub.w     d3,d1           ; U=b-Y
    move.w    d1,&AU          ; AU=U
    move.w    d0,d1           ; V=r
    sub.w     d3,d1           ; V=r-Y
    move.w    d1,&AV          ; AV=V

```

```

    endm

```

- 755 -

Engineering:KlicsCode:CompPict:Color2.a

```

if &TYPE('seg')='UNDEFINED' then
seg      &seg
endif

```

```

RGB2YUV2    FUNC    EXPORT
.
    link      a6,#0          ; no local variables
    movem.l   d4-d7/a3,-(a7) ; store registers
.
    move.l     $0008(a6),a3    ; pm=map
    move.l     $000C(a6),a0    ; Y=Yc
    move.l     $0010(a6),a1    ; U=Uc
    move.l     $0014(a6),a2    ; V=Vc
    move.l     $0018(a6),d7    ; fend=area
    asl.l      #2,d7           ; fend<=2
    add.l      a3,d7           ; fend+=pm
    move.l     $001C(a6),d4    ; width_b=width
    asl.l      #2,d4           ; width_b<=2
    move.l     $0020(a6),d5    ; inc_b=cols
    asl.l      #2,d5           ; cols<=2
    sub.l      d4,d5           ; inc_b-=width_b
    @do1      move.l   a3,d6    ; rowend=pm
    add.l      d4,d6           ; rowend+=width_b
    @do2      rgb2y     (a3)+,(a0)+ ; rgb2y(pm++,Y++)
    rgb2uv     (a1)+,(a2)+ ; rgb2uv(U++,V++)
    rgb2y     (a3)+,(a0)+ ; rgb2y(pm++,Y++)
    cmpa.l     d6,a3           ; rowend>pm
    blt.s      @do2           ; while
    adda.l     d5,a3           ; pm+=inc_b
    move.l     a3,d6           ; rowend=pm
    add.l      d4,d6           ; rowend+=width_b
    @do3      rgb2y     (a3)+,(a0)+ ; rgb2y(pm++,Y++)
    cmpa.l     d6,a3           ; rowend>pm
    blt.s      @do3           ; while
    adda.l     d5,a3           ; pm+=inc_b
    cmpa.l     d7,a3           ; fend>pm
    blt.w      @do1           ; while -
.
    movem.l   (a7)+,d4-d7/a3   ; restore registers
    unlk      a6              ; remove locals
    rts                          ; return
.
    ENDFUNC

```

```

macro
FETCHY      &AY, &Y, &R, &G, &B
.
    move.l     &AY,&Y          ; Y=*AY++
    add.l      &Y,&R           ; RR+=Y12
    add.l      &Y,&G           ; GG+=Y12
    add.l      &Y,&B           ; BB+=Y12
.
endm

```

```

macro
FIXOV      &V, &SP1, &SP2
.
    move.w     &V,&SP1
    clr.b      &SP1
    andi.w     #03FFFF,&SP1
    sne        &SP1
    btst       #13,&SP1
    seq        &SP2
.
endm

```

- 756 -

Engineering:KlacsCode:CompPict:Color2.a

```

or.b      &SP1,&V
and.w     &SP2,&V
swap      &V
move.w    &V,&SP1
clr.b     &SP1
andl.w    *$3FFF,&SP1
sne       &SP1
btst      *13,&SP1
seq       &SP2
or.b      &SP1,&V
and.w     &SP2,&V
swap      &V
endm

-----
macro
OVERFLOW    &A, &B, &SP1, &SP2
.
.   move.l    $SFF00FF00,&SP1      ; sp1=mask
.   move.l    &A,&SP2              ; sp2=ovov (A)
.   and.l     &SP1,&SP2            ; sp2=o0o0 (A)
.   lsr.l     *8,&SP2              ; sp2=0o0o (A)
.   and.l     &B,&SP1              ; sp1=o0o0 (B)
.   or.l      &SP2,&SP1            ; sp1=oooo (EABA)
.   move.l    &A,&SP1
.   or.l      &B,&SP1
.   andl.l    $SFF00FF00,&SP1
.   beq.s     0ok                 ; if no overflow
.   clr.w     &SP2                ; AND=0
.   FIXOV     &A,&SP1,&SP2         ; A1 overflow
.   FIXOV     &B,&SP1,&SP2         ; B1 overflow
6ok
.
endm

-----
macro
MKRGB       &R, &G, &B, &ARGB
.
.   lsl.l     *8,&G                ; G=GGG0 (12)
.   or.l      &B,&G                ; G=GBGB (12)
.   move.l    &R,&B                ; B=OR0R (12)
.   swap      &B                  ; B=OR0R (21)
.   move.w    &G,&B                ; B=ORGB (2)
.   swap      &G                  ; G=GBGB (21)
.   move.w    &G,&R                ; R=ORGB (1)
.   move.l    &R,&ARGB             ; *RGB++=rgb (1)
.   move.l    &B,&ARGB             ; *RGB++=rgb (2)
.
endm

-----
macro
DUPVAL      &V0, &V1
.
.   move.w    &V0,&V1              ; v1=v0
.   swap      &V0
.   move.w    &V1,&V0              ; dup v0
.   move.l    &V0,&V1              ; dup v1
.
endm

-----
macro
UV2RGB3     &AU,&AV

```

- 757 -

Engineering:KlicsCode:CompPict:Color2.a

```

d1 - ra, d2 - ga, d3 - ba, d4 - rb, d5 - gb/512, d6 - bb

```

```

move.w    #512,d5          ; d5=512
move.w    &AU,d2           ; U=*AU++
add.w     d2,d2            ; U is 10(16) bits
move.w    d2,d3           ; ba=U
add.w     d3,d2           ; ga=2U
add.w     d3,d2           ; ga=3U
add.w     d5,d3           ; ba+=512
DUPVAL    d3,d6           ; ba=bb=BB
asr.w     #4,d2           ; ga=3U>>4
move.w    &AV,d1          ; V=*AV++
add.w     d1,d2           ; ga+=V
add.w     d1,d1           ; ra*=2
add.w     d5,d1           ; ra+=512
DUPVAL    d1,d4           ; ra=rb=RR
sub.w     d2,d5           ; gb=512-ga
DUPVAL    d5,d2           ; ga=gb=GG

```

```

endm

```

```

-----
if &TYPE('seg')*UNDEFINED' then
seg      &seg
endif

```

```

YUV2RGB2    FUNC    EXPORT

```

```

PS          RECORD    8
pixmap      DS.L      1
Y           DS.L      1
U           DS.L      1
V           DS.L      1
area        DS.L      1
width       DS.L      1
cols        DS.L      1
            ENDR
LS          RECORD    0,DECR
inc         DS.L      1
width       DS.L      1
fend        DS.L      1
count       DS.L      1
LSize       EQU      *
            ENDR

```

```

a0 - Y0, a1 - Y1, a2 - U, a3 - V, a4 - pm0, a5 - pm1
d6..6 - used, d7 - count

```

```

link        a6,#LS.LSize          ; inc, width, fend and rowend are loca
movem.l     d4-d7/a3-a5,-(a7)      ; store registers

```

```

move.l      PS.pixmap(a6),a4       ; pm0=pixmap
move.l      a4,a5                  ; pm1=pm0
move.l      PS.Y(a6),a0            ; Y0=Yc
move.l      a0,a1                  ; Y1=Y0
move.l      PS.U(a6),a2            ; U=Uc
move.l      PS.V(a6),a3            ; V=Vc
move.l      PS.area(a6),d7         ; fend=area
lsl.l       #2,d7                  ; fend<=2
add.l       a4,d7                  ; fend+=pm0
move.l      d7,LS.fend(a6)         ; save fend
move.l      PS.width(a6),d5        ; width=width
move.l      d5,d7                  ; count=width

```

- 758 -

Engineering:KlicsCode:CompPict:Color2.a

```

asr.l      #1,d7                ; count>=1
subq.l     #1,d7                ; count-=1
move.l     d7,PS,width(a6)      ; save width
add.l      d5,d5                ; width*=2
add.l      d5,a1                ; Y1+=width
add.l      d5,d5                ; width*=2
move.l     d5,LS,width(a6)      ; save width
move.l     PS,cols(a6),d4       ; inc=cols
lsl.l      #2,d4                ; inc<=2
add.l      d4,a5                ; pml+=inc
add.l      d4,d4                ; cols*=2
sub.l      d5,d4                ; inc now 2*cols-width bytes
move.l     d4,LS,inc(a6)        ; save inc
@do
UV2RGB3    (a2)+,(a3)-          ; uv2rgb('U++','V++')
FETCHY     (a0)+,d0,d1,d2,d3    ; add Ya to RGB values
FETCHY     (a1)+,d0,d4,d5,d6    ; add Yb to RGB values
move.w     #0xFFFF,d0          ; d0=mask
lsl.l      #2,d1                ; d1 8(16) bits
and.w      d0,d1                ; d1 masked
lsl.l      #2,d2                ; d2 8(16) bits
and.w      d0,d2                ; d2 masked
lsl.l      #2,d3                ; d3 8(16) bits
and.w      d0,d3                ; d3 masked
lsl.l      #2,d4                ; d4 8(16) bits
and.w      d0,d4                ; d4 masked
lsl.l      #2,d5                ; d5 8(16) bits
and.w      d0,d5                ; d5 masked
lsl.l      #2,d6                ; d6 8(16) bits
and.w      d0,d6                ; d6 masked
move.l     d1,d0
or.l       d2,d0
or.l       d3,d0
or.l       d4,d0
or.l       d5,d0
or.l       d6,d0
andi.l     #0xFFFF,d0
@over
bne.s      @over                ; if overflow
@ok
MRGBB      d1,d2,d3,(a4)+        ; save RGBa
MRGBB      d4,d5,d6,(a5)+        ; save RGBb
dbf        d7,@do                ; while
adda.l     LS,inc(a6),a4          ; pm0+=inc
adda.l     LS,inc(a6),a5          ; pml+=inc
adda.l     LS,width(a6),a0        ; Y0+=width
exg.l      a0,a1                 ; Y1<->Y0
move.l     PS,width(a6),d7        ; count=width
cmpa.l     LS,fend(a6),a4         ; pm0<fend
bit.w      @do                    ; while
movem.l    (a7)+,d4-d7/a3-a5      ; restore registers
unlk       a6                    ; remove locals
rts        ; return
@over
move.l     d7,LS,count(a6)        ; save count
clr.w      d7                    ; AND=0
FIXOV      d1,d0,d7                ; A overflow
FIXOV      d2,d0,d7                ; B overflow
FIXOV      d3,d0,d7                ; A overflow
FIXOV      d4,d0,d7                ; B overflow
FIXOV      d5,d0,d7                ; A overflow
FIXOV      d6,d0,d7                ; B overflow
move.l     LS,count(a6),d7        ; restore count
bra        @ok

```

- 759 -

Engineering:Kl:csCode:CompPict:Color2.a

```

ENDFUNC
-----
if &TYPE('seg')='UNDEFINED' then
seg      &seg
endif

GREY2Y   FUNC      EXPORT
.
PS        RECORD      8
pixmap   DS.L         1
Y         DS.L         1
area      DS.L         1
width     DS.L         1
cols      DS.L         1
ENDR

.
d0 - vvvv, d1 - v0v1, d2 - v2v3, d3 - xor, d4 - width, d5 - inc, d6 - rowend,
a0 - pm, a1 - Y
.
      link          a6,#0                ; no local variables
      movem.l       d4-d7,-(a7)          ; store registers

      move.l        PS,pixmap(a6),a0     ; pm=pixmap
      move.l        PS,Y(a6),a1          ; Y=Yc
      move.l        PS,area(a6),d7       ; fend=area
      add.l         a0,d7                 ; fend=spm
      move.l        PS,width(a6),d4      ; width_b=width
      move.l        PS,cols(a6),d5       ; inc_b=cols
      sub.l         d4,d5                 ; inc_b=width_b
      move.l        #57F7F7F7,d3         ; xor=57F7F7F7
&d01    move.l        a0,d6               ; rowend=pm
      add.l         d4,d6                 ; rowend=width_b
&d02    move.l        (a0)+,d0            ; vvvv=pm
      eor.l         d3,d0                ; vvvv is signed
      move.w        d0,d2                 ; d2=v2v3
      asr.w         #6,d2                 ; d2=v2 (10 bits)
      swap          d2                   ; d2=v2??-
      move.b        d0,d2                 ; d2=v2v3
      ext.w         d2                    ; v3 extended
      lsl.w         #2,d2                 ; d2=v2v3 (10 bits)
      swap          d0                    ; d0=v0v1
      move.w        d0,d1                 ; d1=v0v1
      asr.w         #6,d1                 ; d1=v0 (10 bits)
      swap          d1                    ; d1=v0??
      move.b        d0,d1                 ; d1=v0v1
      ext.w         d1                    ; v1 extended
      lsl.w         #2,d1                 ; d1=v0v1 (10 bits)
      move.l        d1,(a1)+              ; *Y=d1
      move.l        d2,(a1)+              ; *Y=d2
      cmpa.l        d6,a0                 ; rowend>pm
      blt.s         &d02                  ; while
      adda.l        d5,a0                 ; pm+=inc_b
      cmpa.l        d7,a0                 ; fend>pm
      blt.s         &d01                  ; while

      movem.l       (a7)+,d4-d7          ; restore registers
      unlk          a6                   ; remove locals
      rts                    ; return

ENDFUNC
-----
if &TYPE('seg')='UNDEFINED' then
seg      &seg
endif

```


- 760 -

Engineering:KlacsCode:CompPict:Color2.a

```

endif
Y2GREY FUNC EXPORT
PS RECORD 3
pixmap DS.L 1
Y DS.L 1
height DS.L 1
width DS.L 1
cols DS.L 1
ENDR

c0- spare, d1 - v43, d2 - v21, d3 - spare, d4 - width, d5 - inc, d6 - count, d
a0 - pm, a1 - Y

link a6, #0 ; no local variables
movem.l d4-d7, -(a7) ; store registers

move.l PS.pixmap(a6), a0 ; pm=pixmap
move.l PS.Y(a6), a1 ; Y=Yc
move.l PS.height(a6), d7 ; long height
subq.l #1, d7 ; height-=1
move.l PS.width(a6), d4 ; long width
move.l PS.cols(a6), d5 ; long inc=cols
sub.l d4, d5 ; inc-=width
lsl.l #2, d4 ; width>=2 (read 4 values)
subq.l #1, d4 ; width-=1
@dc1 move.l d4, d6 ; count=width
@dc2 move.l (a1)+, d0 ; d0=x4x3
move.l (a1)+, d1 ; d1=x2x1
move.l #01FF01FF, d2 ; d2=511
move.l d2, d3 ; d3=511
sub.l d0, d2 ; unsigned d2
sub.l d1, d3 ; unsigned d3
lsl.l #2, d2
lsl.l #2, d3
move.l d2, d0
or.l d3, d0
andi.l #03F003F00, d0
bne.s @over ; if no overflow
@ok lsl.l #8, d3 ; d3=0210
lsl.l #8, d2 ; d2=0430
lsl.l #8, d3 ; d3=0021
lsl.l #8, d2 ; d2=4300
or.l d3, d2 ; d2=4321
move.l d2, (a0)+ ; *pm=d2
dbf d6, @dc2 ; while -1!--count
adda.l d5, a0 ; pm+=inc_b
dbf d7, @dc1 ; while -1!--height

movem.l (a7)+, d4-d7 ; restore registers
unlk a6 ; remove locals
rts ; return
@over clr.l d1 ; AND=0
FIXOV d2, d0, d1 ; A overflow
FIXOV d3, d0, d1 ; B overflow
bra.s @ok

ENDFUNC
-----
macro
GGG &V, &SP1, &SP2, &AV

```

- 761 -

Engineering:KlicsCode:CompPict:Color3.2

```

move.l    &V,&SP2           ; SP2=0102
lsl.l     &8,&SP2           ; SP2=1020
cr.l      &V,&SP2           ; SP2=1122
move.l    &V,&SP1          ; SP1=0102
swap      &SP1              ; SP1=C201
move.w    &SP2,&SP1         ; SP1=0222
swap      &SP2              ; SP2=2211
move.w    &SP2,&V           ; V=0111
move.l    &V,&AV            ; *pm=V
move.l    &SP1,&AV          ; *pm=SP1

endm

-----
if &TYPE('seg')='UNDEFINED' then
seg      &seg
endif

Y2GGG    FUNC      EXPORT
.
PS        RECORD      8
pixmap    DS.L        1
Y          DS.L        1
lines     DS.L        1
width     DS.L        1
cols      DS.L        1
ENDR

.
d0 - v, d4 - width, d5 - inc, d6 - count, d7 - lines
a0 - pm, a1 - Y

link      a6,#0             ; no local variables
movem.l   d4-d7,-(a7)       ; store registers

.
move.l    PS,pixmap(a6),a0   ; pm=pixmap
move.l    PS,Y(a6),a1        ; Y=Yc
move.l    PS,lines(a6),d7    ; long lines
subq.l    #1,d7              ; lines--1
move.l    PS,width(a6),d4    ; long width
move.l    PS,cols(a6),d5     ; inc=cols
sub.l     d4,d5              ; inc--width
lsl.l     #2,d5              ; inc (bytes)
lsr.l     #2,d4              ; width>>2
subq.l    #1,d4              ; width--1
9dc1      move.l    d4,d6     ; count=width
9dc2      move.l    (a1)+,d0   ; d0=x1x2 (10 bits signed)
          move.l    (a1)+,d1   ; d1=x3x4 (10 bits)
          move.l    $02000200,d3 ; d3=plus
          add.l     d3,d0       ; d0=x1x2 (unsigned)
          add.l     d3,d1       ; d1=x3x4 (unsigned)
          lsr.l     #2,d0       ; d0=x1x2 (10,8 bits)
          lsr.l     #2,d1       ; d1=x3x4 (10,8 bits)
          move.w    #03FFF,d2   ; d2=mask
          and.w     d2,d0       ; mask d0
          and.w     d2,d1       ; mask d1
          move.l    d0,d2
          or.l      d1,d2
          andi.l    #0FFF00,d2
          bne.s     9over      ; if no overflow
9ok        GGG      d0,d2,d3,(a0)+
          GGG      d1,d2,d3,(a0)+
          dbf      d6,9dc2      ; while -1!--count
          adda.l    d5,a0       ; pm+=inc_b
          dbf      d7,9dc1      ; while -1!--lines

```

- 762 -

- Engineering:KlidsCode:CompPict:Color2.a

```

movem.l    (a7)+,d4-d7      ; restore registers
unlk       a6               ; remove locals
rts        ; return
cover      d3               ; AND=0
FIXOV      d0,d2,d3         ; A overflow
FIXOV      d1,d2,d3         ; B overflow
bra.w      20k
ENDFUNC
-----
macro
MXRGB2     &R, &G, &B, &ARGB, &ROW, &XX

    lsl.l   &B,&G           ; G=G0G0 (12)
    or.l    &B,&G           ; G=GBGB (12)
    move.l  &R,&B           ; B=0R0R (12)
    swap    &B             ; B=0R0R (21)
    move.w  &G,&B           ; B=0RGB (2)
    swap    &G             ; G=GBGB (21)
    move.w  &G,&R           ; R=0RGB (1)

    andi.l  *FFFFFFFE,&R    ; 7 bits for interpolation
    andi.l  *FFFFFFFE,&B    ; 7 bits for interpolation

    move.l  &R,&G           ; G=RGB(1)
    add.l   &B,&G           ; G+=RGB(2)
    lsr.l   &1,&G           ; G/=2

    move.l  &B,&XX          ; XX=RGB(2)
    sub.l   &R,&XX          ; XX-=RGB(1)
    lsr.l   &1,&XX          ; XX/=2
    add.l   &B,&XX          ; XX+=B

    move.l  &R,(&ARGB)+     ; *RGB++=rgb (1)
    move.l  &G,(&ARGB)+     ; *RGB++=rgb (1.5)
    move.l  &B,(&ARGB)+     ; *RGB++=rgb (2)
    move.l  &B,(&ARGB)+     ; *RGB++=rgb (2.5)

    add.l   &ROW,&ARGB
    sub.l   #16,&ARGB

    move.l  &R,(&ARGB)+     ; *RGB++=rgb (1)
    move.l  &G,(&ARGB)+     ; *RGB++=rgb (1.5)
    move.l  &B,(&ARGB)+     ; *RGB++=rgb (2)
    move.l  &B,(&ARGB)+     ; *RGB++=rgb (2.5)

    sub.l   &ROW,&ARGB

endm
-----
if &TYPE('seg')='UNDEFINED' then
seg      &seg
endif

YUV2RGB3  FUNC      EXPORT
.
PS         RECORD      8
pixmap     DS.L        1
Y          DS.L        1
U          DS.L        1
V          DS.L        1
area       DS.L        1

```

- 763 -

Engineering:KlicsCode:CompPict:Color2.a

```

width DS.L 1
cols DS.L 1
ENDR
.
LS RECORD 0,DECR
inc DS.L 1
width DS.L 1
fend DS.L 1
count DS.L 1
row DS.L 1
LSize EQU .
ENDR
.
a0 = Y0, a1 = Y1, a2 = U, a3 = V, a4 = pm0, a5 = pm1
d0..6 = used, d7 = count

link a6,LS,LSize ; inc. width, fend and rowend are loca
movem.l d4-d7/a3-a5,-(a7) ; store registers

move.l PS.pixmap(a6),a4 ; pm0=pixmap
move.l a4,a5 ; pm1=pm0
move.l PS.Y(a6),a0 ; Y0=Yc
move.l a0,a1 ; Y1=Y0
move.l PS.U(a6),a2 ; U=Uc
move.l PS.V(a6),a3 ; V=Vc
move.l PS.area(a6),d7 ; fend=area
lsl.l #2,d7 ; fend<<=2
add.l a4,d7 ; fend+=pm0
move.l d7,LS.fend(a6) ; save fend
move.l PS.width(a6),d5 ; width=width
move.l d5,d7 ; count=width
asr.l #1,d7 ; count>>=1
subq.l #1,d7 ; count-=1
move.l d7,PS.width(a6) ; save width
add.l d5,d5 ; width*=2
add.l d5,a1 ; Y1+=width
add.l d5,d5 ; width*=2
move.l d5,LS.width(a6) ; save width
move.l PS.cols(a6),d4 ; inc=cols
lsl.l #2,d4 ; inc<<=2
move.l d4,LS.row(a6) ; NEW save row
add.l d4,a5 ; pm1+=inc
add.l d4,a5 ; NEW pm1+=inc
add.l d4,d4 ; cols*=2
sub.l d5,d4 ; inc now 4*cols-width bytes
sub.l d5,d4 ; NEW inc now 4*cols-width bytes (wid
move.l d4,LS.inc(a6) ; save inc
@do UV2RGB3 (a2)+,(a3)- ; uv2rgb(*U++,*V++)

FETCHY (a0)+,d0,d1,d2,d3 ; add Ya to RGB values
FETCHY (a1)+,d0,d4,d5,d6 ; add Yb to RGB values

move.w #03FFF,d0 ; d0=mask
lsl.l #2,d1 ; d1 8(16) bits
and.w d0,d1 ; d1 masked
lsl.l #2,d2 ; d2 8(16) bits
and.w d0,d2 ; d2 masked
lsl.l #2,d3 ; d3 8(16) bits
and.w d0,d3 ; d3 masked
lsl.l #2,d4 ; d4 8(16) bits
and.w d0,d4 ; d4 masked
lsl.l #2,d5 ; d5 8(16) bits

```

- 764 -

Engineering:KlacsCode:CompPict:ColorB.a

```

and.w    d0,d5          ; d5 masked
lsl.l    #2,d5          ; d6 8(16) bits
and.w    d0,d5          ; d6 masked

move.l    d1,d0
cr.l     d2,d0
cr.l     d3,d0
cr.l     d4,d0
cr.l     d5,d0
cr.l     d6,d0
andi.l    $5FF00FF0,d0
bne.w     @over          ; if overflow

;ok MKRGB2 d1,d2,d3,a4,LS.row(a6),d0 ; *NEW save RGBa
MKRGB2 d4,d5,d6,a5,LS.row(a6),d0 ; *NEW save RGBb
dbf      d7,@do          ; while
adda.l    LS.inc(a6),a4   ; pm0+=inc
adda.l    LS.inc(a6),a5   ; pm1+=inc
adda.l    LS.width(a6),a0 ; Y0+=width
exg.l     a0,a1           ; Y1<->Y0
move.l    PS.width(a6),d7 ; count=width
cmpa.l    LS.fend(a6),a4  ; pm0<fend
blt.w     @do            ; while

movem.l    (a7)+,d4-d7/a3-a5 ; restore registers
unlk      a6              ; remove locals
rts        ; return
;over move.l    d7,LS.count(a6) ; save count
; AND=0
; A overflow
; B overflow
; A overflow
; B overflow
; A overflow
; B overflow
; restore count
move.l    LS.count(a6),d7
bra       @ok

ENDFUNC
-----
macro
FETCHY2    &AY, &Y, &R, &G, &B
; Y
move.l    &AY,&Y
asr.w     #2,&Y
swap
asr.w     #2,&Y
swap
add.l     &Y,&R
add.l     &Y,&G
add.l     &Y,&B
; Y is -128 to -127
; RED, Get (Y- 2V + 512) for Red = (Y +
; GREEN, Get (Y + (512 - (6U/16)) - V)
; BLUE, Get (Y + (2U + 512) for Blue = (
endm
-----
macro
UV2RGB4    &AU, &AV
; U
move.w     &AU,d2
and.w     #503FF,d2
move.l     (a6,d2.w*8),d3
move.l     d3,d6
move.l     4(a6,d2.w*8),d5
move.w     &AV,d1
; V
; BLUE, Get (2U + 512)/4 for Blue = (Y +
; Dup for second pair
; GREEN, Get (512 - (6U/16))/4 for Gree

```

- 765 -

Engineering:KlicsCode:CompPict:Color3.a

```

move.w    d1,d4
asr.w     #2,d1
sub.w     d1,d5           ; GREEN, Get (512 - (EU/16) - V)/4 for
move.w     d5,d2
swap      d5
move.w     d2,d5
move.l     d5,d2           ; Dup for second pair

and.w     #03FF,d4
move.l     (a6,d4.w*8),d4   ; RED, Get (2V + 512)/4 for Red = (Y -
move.l     d4,d1

```

```

endm
-----

```

MKRGB2SUB FUNC EXPORT

```

MKRGB2    d1,d2,d3,a4,d7,d0 ; *NEW save RGBa
MKRGB2    d4,d5,d6,a5,d7,d0 ; *NEW save RGBb
rts

```

ENDFUNC

OVERSUB FUNC EXPORT

```

move.l     d1,d0
or.l       d2,d0
or.l       d3,d0
or.l       d4,d0
or.l       d5,d0
or.l       d6,d0
andi.l     #5FFF00FF,d0
bne.s      $over          ; if overflow
$ok        rts
$over      move.l     d7,-(sp)      ; save count
           clr.w       d7          ; AND=0
           FIXOV      d1,d0,d7     ; A overflow
           FIXOV      d2,d0,d7     ; B overflow
           FIXOV      d3,d0,d7     ; A overflow
           FIXOV      d4,d0,d7     ; B overflow
           FIXOV      d5,d0,d7     ; A overflow
           FIXOV      d6,d0,d7     ; B overflow
           move.l     (sp)+,d7     ; restore count
           bra        $ok

```

ENDFUNC

UV2RGB4SUB FUNC EXPORT

```

UV2RGB4    (a2)+,(a3)+       ; uv2rgb(*U++,*V++)
rts

```

ENDFUNC

FETCHY2SUB FUNC EXPORT

```

FETCHY2    (a0)+,d0,d1,d2,d3 ; add Ya to RGB values
FETCHY2    (a1)+,d0,d4,d5,d6 ; add Yb to RGB values
rts

```

ENDFUNC

```

if $TYPE('seg') < 'UNDEFINED' then

```

- 766 -

Engineering:KlmcCode:CompPact:Color2.a

```

        seg      6seg
        endif

YUV2RGB5  FUNC      EXPORT
.
PS        RECORD      3
Table     DS.L        1
pixmap    DS.L        1
Y         DS.L        1
U         DS.L        1
V         DS.L        1
area      DS.L        1
width     DS.L        1
cols      DS.L        1
        ENDR
.
LS        RECORD      0,DECR
inc        DS.L        1
width     DS.L        1
fend      DS.L        1
count     DS.L        1
row       DS.L        1
LSize     EQU         -
        ENDR
.
        a0 - Y0, a1 - Y1, a2 - U, a3 - V, a4 - pm0, a5 - pm1
        d0..6 - used, d7 - count
.
        link      a6,*LS.LSize      ; inc. width, fend and rowend are loca
        movem.l   d4-d7/a3-a5,-(a7) ; store registers
.
        move.l    PS.pixmap(a6),a4   ; pm0=pixmap
        move.l    a4,a5              ; pm1=pm0
        move.l    PS.Y(a6),a0        ; Y0=Yc
        move.l    a0,a1              ; Y1=Y0
        move.l    PS.U(a6),a2        ; U=Uc
        move.l    PS.V(a6),a3        ; V=Vc
        move.l    PS.area(a6),d7     ; fend=area
        lsl.l     #2,d7              ; fend<==2
        add.l     a4,d7              ; fend==pm0
        move.l    d7,LS.fend(a6)     ; save fend
        move.l    PS.width(a6),d5    ; width=width
        move.l    d5,d7              ; count=width
        asr.l     #1,d7              ; count>>=1
        subq.l    #1,d7              ; count-=1
        move.l    d7,PS.width(a6)    ; save width
.
        add.l     d5,d5              ; width*=2
        add.l     d5,a1              ; Y1+=width
        add.l     d5,d5              ; width*=2
        move.l    d5,LS.width(a6)    ; save width
        move.l    PS.cols(a6),d4     ; inc=cols
        lsl.l     #2,d4              ; inc<==2
        move.l    d4,LS.row(a6)      ; *NEW save row
        add.l     d4,a5              ; pm1+=inc
        add.l     d4,a5              ; *NEW pm1+=inc
        add.l     d4,d4              ; cols*=2
        add.l     d4,d4              ; *NEW cols*=2
        sub.l     d5,d4              ; inc now 4*cols-width bytes
        sub.l     d5,d4              ; *NEW inc now 4*cols-width bytes (wid
        move.l    d4,LS.inc(a6)      ; save inc
.
        add      move.l    d7,-(sp)

```

- 767 -

Engineering:KilicsCode:CompPict:Color2.a

```

move.l    a6,-(sp)
move.l    LS.row(a6),d7
move.l    PS.Table(a6),a6
UV2RGB4    (a2)+,(a3)+          ; uv2rgb(*U--,*V--)

FETCHY1    (a0)+,d0,d1,d2,d3      ; add Ya to RGB values
FETCHY2    (a1)+,d0,d4,d5,d6      ; add Yb to RGB values

move.l    d1,d0
or.l      d2,d0
or.l      d3,d0
or.l      d4,d0
or.l      d5,d0
or.l      d6,d0
andi.l    *$FF00FF00,d0
bne.w     @over          ; if overflow

@ok        MKRGB2    d1,d2,d3,a4,d7,d0 ; *NEW save RGBa
           MKRGB2    d4,d5,d6,a5,d7,d0 ; *NEW save RGBb
           move.l    (sp)+,a6
           move.l    (sp)+,d7

           dbf      d7,@do          ; while

           adda.l    LS.inc(a6),a4      ; pm0--inc
           adda.l    LS.inc(a6),a5      ; pm1--inc
           adda.l    LS.width(a6),a0    ; Y0--width
           exg.l     a0,a1              ; Y1<--Y0
           move.l    PS.width(a6),d7    ; count=width
           cmpa.l    LS.fend(a6),a4     ; pm0<fend
           blt.s     @do               ; while

           movem.l   (a7)+,d4-d7/a3-a5  ; restore registers
           unlk      a6                 ; remove locals
           rts                    ; return
@over      move.l    d7,LS.count(a6)    ; save count
           clr.w     d7                 ; AND=0
           FIXOV     d1,d0,d7           ; A overflow
           FIXOV     d2,d0,d7           ; B overflow
           FIXOV     d3,d0,d7           ; A overflow
           FIXOV     d4,d0,d7           ; B overflow
           FIXOV     d5,d0,d7           ; A overflow
           FIXOV     d6,d0,d7           ; B overflow
           move.l    LS.count(a6),d7    ; restore count
           bra       @ok

           ENDFUNC
           -----
           END

```


- 768 -

Engineering:KLICSCode:CompFact:Clut.c

```

.....
*   © Copyright 1993 KLICS Limited
*   All rights reserved.
*   Written by: Adrian Lewis
*
.....
/*
  Analyse CLUT setup and pick appropriate
  YUV->RGB converter/display driver. Create
  any tables necessary.
*/

#include <QuickDraw.h>
#include <Memory.h>

#define Y_LEVELS    64
#define UV_LEVELS   16

#define absv(v) ((v)<0?-(v):(v))
#define NewPointer(ptr,type,size) \
    saveZone=GetZone(); \
    SetZone(SystemZone()); \
    if (nil==(ptr=(type)NewPtr(size))) { \
        SetZone(ApplicZone()); \
        if (nil==(ptr=(type)NewPtr(size))) { \
            SetZone(saveZone); \
            return(MemoryError()); \
        } \
    } \
    SetZone(saveZone);

typedef struct {
    char    y, u, v;
} YUV_Clut;

/*
unsigned char *
ColourClut(CTabHandle clut)
{
    int      size, y, u, v, r, g, b, i;
    unsigned char *table;
    YUV_Clut  *yuv_clut;

    size=((clut)->ctSize);
    table=(unsigned char *)NewPtr(Y_LEVELS*UV_LEVELS*UV_LEVELS);
    yuv_clut=(YUV_Clut *)NewPtr(size*sizeof(YUV_Clut));

    for(i=0;i<size;i++) {
        r=((clut)->ctTable[i].rgb.red>>8)-128;
        g=((clut)->ctTable[i].rgb.green>>8)-128;
        b=((clut)->ctTable[i].rgb.blue>>8)-128;

        yuv_clut[i].y= (306*r + 601*g + 117*b)>>10;
        yuv_clut[i].u= (512*r - 429*g - 83*b)>>10;
        yuv_clut[i].v= (-173*r - 339*g + 512*b)>>10;
    }
    for(y=-Y_LEVELS/2;y<Y_LEVELS/2-1;y++)
        for(u=-UV_LEVELS/2;u<UV_LEVELS/2-1;u++)
            for(v=-UV_LEVELS/2;v<UV_LEVELS/2-1;v++) {
                int      index,error,error2,points, Y, U, V;

```

- 769 -

Engineering:KlicsCode:CompPict:Clut.c

```

Y=y<<4;
U=u<<5;
V=v<<5;

index=0;
error=131072;
error2=131072;
points=0;
for(i=0;i<=size;i++) {
    int pts=0, err=0;

    if (yuv_clut[i].y>=Y && yuv_clut[i].y<Y+16)
        pts+=1;
    err+=absv(yuv_clut[i].y-Y);

    if (yuv_clut[i].u>=U && yuv_clut[i].u<U+32)
        pts+=1;
    err+=absv(yuv_clut[i].u-U);

    if (yuv_clut[i].v>=V && yuv_clut[i].v<V+32)
        pts+=1;
    err+=absv(yuv_clut[i].v-V);

    if (pts>points || (pts==points && err<error)) {
        error=err;
        index=i;
        points=pts;
    }
}
i=((y<0x1F)<<8)|((u<0xF)<<4)|(v<0xF);
table[i]=(unsigned char)index;
}
DisposePtr((Ptr)yuv_clut);
return table;
}

typedef union {
    long pixel;
    unsigned char rgb[4];
} Pixel;
/*
unsigned long *
ColourClut(CTabHandle clut)
{
    long size, y, u, v, r, g, b, ro, go, bo, i;
    Pixel *table;

    size=(clut)->ctSize;
    table=(Pixel *)NewPtr(Y_LEVELS*UV_LEVELS*UV_LEVELS*sizeof(long));

    for(y=-Y_LEVELS/2;y<Y_LEVELS/2-1;y++)
    for(u=-UV_LEVELS/2;u<UV_LEVELS/2-1;u++)
    for(v=-UV_LEVELS/2;v<UV_LEVELS/2-1;v++) {
        Pixel px;
        long base, dith;

        r = 32768L + ((y<<9) + 1436L*u <<2);
        g = 32768L + ((y<<9) - 731L*u - 352L*v <<2);
        b = 32768L + ((y<<9) + 1815L*v <<2);

        r=r<0?0:r>65534?65534:r;
        g=g<0?0:g>65534?65534:g;
        b=b<0?0:b>65534?65534:b;
    }
}

```

- 770 -

➤ Eng:neering:KitsCode:CompPic:Clus.c

```

r=r\13107; r=r/13107;
go=g\13107; g=g/13107;
bo=b\13107; b=b/13107;

base=215-(35*r-6*g-b);

dith=base-(ro>2621736:0)-(gc>786376:0)-(bo>1048471:0);
px.rgb[0]=dith==215?255:dith;

dith=base-(ro>5242736:0)-(go>1048476:0)-(bo>262171:0);
px.rgb[1]=dith==215?255:dith;

dith=base-(ro>7863736:0)-(go>262176:0)-(bc>524271:0);
px.rgb[2]=dith==215?255:dith;

dith=base-(ro>10484736:0)-(go>524276:0)-(bo>786371:0);
px.rgb[3]=dith==215?255:dith;

i=((y&0x3F)<<8)|((u&0xF)<<4)|(v&0xF);

table[i].pixel=px.pixel;

return (unsigned long*)table;
},

typedef struct {
    long    red, green, blue;
} RGBError;

OSErr ColourClut(Pixel **table)
{
    long    y, u, v, r, g, b, i;
    RGBError    *err;
    THz        saveZone;

    NewPointer(*table, Pixel*, Y_LEVELS*UV_LEVELS*UV_LEVELS*sizeof(long)); /* 64k ta
    NewPointer(err, RGBError*, Y_LEVELS*UV_LEVELS*UV_LEVELS*sizeof(RGBError));

    for(i=0; i<4; i++)
    for(y=-Y_LEVELS/2; y<Y_LEVELS/2; y++)
    for(u=-UV_LEVELS/2; u<UV_LEVELS/2; u++)
    for(v=-UV_LEVELS/2; v<UV_LEVELS/2; v++) {
        RGBColor    src, dst;
        long        index, in;

        index=((y&0x3F)<<8)|((u&0xF)<<4)|(v&0xF);

        r = 32768L + ((y<<9) + (1436L*u) <<2);
        g = 32768L + ((y<<9) - (731L*u) - (352L*v) <<2);
        b = 32768L + ((y<<9) + (1815L*v) <<2);

        if (i>0) {
            r--err[index].red;
            g--err[index].green;
            b--err[index].blue;
        }

        src.red=r<0?0:r>65534?65534:r;
        src.green=g<0?0:g>65534?65534:g;
        src.blue=b<0?0:b>65534?65534:b;

        (*table)[index].rgb[i]=(unsigned char)Color2Index(4src);
    }
}

```

- 771 -

= Engineering:KlacsCode:CompPict:Cluc.c

```

    Index2Color((*table)[index].rgb[i].dst);
    err[index].red=dst.red-src.red;
    err[index].green=dst.green-src.green;
    err[index].blue=dst.blue-src.blue;
}
DisposePtr((Ptr)err);
return(noErr);
)

typedef struct (
    short    pel[2];
) Pix16;

typedef struct (
    unsigned char    pel[4];
) Pix8;

#define YS 64
#define UVS 32

OSErr Colour8(Pix8 **table)
(
    long    y, u, v, r, g, b, i;
    RGBError *err;
    THz    saveZone;

    NewPointer(*table, Pix8*, YS*UVS*UVS*sizeof(Pix8)); /* 128k table */
    NewPointer(err, RGBError*, YS*UVS*UVS*sizeof(RGBError));

    for(i=0; i<4; i++)
        for(y=-YS/2; y<YS/2; y++)
            for(u=-UVS/2; u<UVS/2; u++)
                for(v=-UVS/2; v<UVS/2; v++) (
                    RGBColor    src, dst;
                    long    index;

                    index=(y<<10)|((u<0x1F)<<5)|((v<0x1F));

                    r = 32768L + ((y<<10) + (1436L*u) <<1);
                    g = 32768L - ((y<<10) - (731L*u) - (352L*v) <<1);
                    b = 32768L - ((y<<10) + (1815L*v) <<1);

                    if (i>0) {
                        r--err[32768+index].red;
                        g--err[32768+index].green;
                        b--err[32768+index].blue;
                    }

                    src.red=r<0?0:r>65534?65534:r;
                    src.green=g<0?0:g>65534?65534:g;
                    src.blue=b<0?0:b>65534?65534:b;

                    (*table)[32768+index].pel[i]=(unsigned char)Color2Index(&src);
                    Index2Color((*table)[32768+index].pel[i].dst);

                    err[32768+index].red=dst.red-src.red;
                    err[32768+index].green=dst.green-src.green;
                    err[32768+index].blue=dst.blue-src.blue;
                )
    DisposePtr((Ptr)err);
    return(noErr);
)

```

- 772 -

Engineering:KlitsCode:CompPict:Clut.c

OSErr Colour16(Pix16 **table)

```

long    y, u, v, r, g, b;
RGBError *err;
THz     saveZone;

NewPointer((table, Pix16*, YS*UVS*UVS*sizeof(Pix16)); /* 128k table */
NewPointer((err, RGBError*, YS*UVS*UVS*sizeof(RGBError));

for(i=0; i<2; i++)
for(y=-YS/2; y<YS/2; y++)
for(u=-UVS/2; u<UVS/2; u++)
for(v=-UVS/2; v<UVS/2; v++) {
    RGBColor  src, dst;
    long      index;

    index=(y<<10)|((u<0x1F)<<5)|(v<0x1F);

    r = 32768L + ((y<<10) + (1436L*u) <<1);
    g = 32768L + ((y<<10) + (731L*u) - (352L*v) <<1);
    b = 32768L + ((y<<10) + (1915L*v) <<1);

    if (i>0) {
        r=err[32768+index].red;
        g=err[32768+index].green;
        b=err[32768+index].blue;
    }

    src.red=r<0?0:r>65534?65534:r;
    src.green=g<0?0:g>65534?65534:g;
    src.blue=b<0?0:b>65534?65534:b;

    dst.red= src.red&0xF800;
    dst.green= src.green&0xF800;
    dst.blue= src.blue&0xF800;

    (*table)[32768+index].pel[i]=(dst.red>>1)|(dst.green>>6)|(dst.blue>>11);

    err[32768+index].red=dst.red-src.red;
    err[32768+index].green=dst.green-src.green;
    err[32768+index].blue=dst.blue-src.blue;
}
DisposePtr((Ptr)err);
return(noErr);
}

```

```

Boolean
GreyClut(CTabHandle clut)
{

```

```

    Boolean result=true;
    int      i, size;

    size=(*clut)->ctSize;
    for(i=0; i<size && result; i++) {
        int      r,g,b;

        r=(*clut)->ctTable[i].rgb.red;
        g=(*clut)->ctTable[i].rgb.green;
        b=(*clut)->ctTable[i].rgb.blue;

        result=(r==g && g==b);
    }
}

```

- 773 -

Engineering:KlicsCode:CompPict:Clut.c

return result;

- 774 -

Engineering:KLICSCode:CompPict:Bits3.h

```

.....
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
...../
/*
  Bits3.h: fast bit read/write definitions

  buf_use      define static variables
  buf_winit    initialise vars for write
  buf_rinit    initialise vars for read
  buf_set      set current bit
  buf_get      get current bit
  buf_winc     increment write buffer
  buf_rinc     increment read buffer
  buf_size     fullness of buffer in bytes
  buf_flush    flush buffer

  User defined macro/function buf_over must be defined in case of buffer overflow
*/

typedef struct (
  unsigned long   *buf;
  union (
    unsigned long  mask;
    long           bno;
  ) index;
  unsigned long   *ptr, data, size;
) Buffer, *Buf;

#define buf_winit(buf) \
  buf->index.mask=0x80000000; \
  buf->ptr=&buf->buf[0]; \
  buf->data=0;

#define buf_rinit(buf) \
  buf->index.bno=0; \
  buf->ptr=&buf->buf[0];

#define buf_set(buf) \
  buf->data |= buf->index.mask;

#define buf_get(buf) \
  0!=(buf->data & (1<<buf->index.bno) )

#define buf_winc(buf) \
  if (buf->index.mask==1) { \
    *buf->ptr=buf->data; \
    buf->data=0; \
    buf->index.mask=0x80000000; \
    buf->ptr++; \
  } else buf->index.mask >>= 1;

#define buf_rinc(buf) \
  if (--(buf->index.bno)<0) { \
    buf->data=*buf->ptr++; \
    buf->index.bno=31; \
  };

/* buf_size only valid after buf_flush */

```

- 775 -

= Engineering:KlicsCode:CompPict:Bits3.h

```
*define buf_size(buf) \
    (unsigned char *)buf->ptr-(unsigned char *)&buf->buf[0]

#define buf_flush(buf) \
    if (buf->index.mask!=0x80000000) { \
        buf->data |= buf->index.mask-1; \
        *buf->ptr=buf->data; \
        buf->ptr++; \
    }
```


- 776 -

Engineering:KLICSCode:CompPict:Bits3.2

```

.....
*   © Copyright 1993 KLICS Limited
*   All rights reserved.
*
*   Written by: Adrian Lewis
*
.....

```

```

.....
*   63000 Bit buffer code (Bits2.h)
*
.....

```

```

.....
*   Macros:

```

```

*   buf_winit    &ptr, &data, &mask, &buf
*   buf_rinit    &ptr, &bno, &buf
*   buf_set      &data, &mask
*   buf_get      &data, &bno
*   buf_winc     &ptr, &data, &mask
*   buf_rinc     &ptr, &data, &index
*   buf_flush    &ptr, &data, &mask

```

```

.....
*   macro

```

```

*   buf_winit    &ptr, &data, &mask, &buf

```

```

*   move.l       $80000000, &mask      ; mask=100...
*   move.l       &buf, &ptr           ; ptr=buf
*   clr.l        &data                ; data=0

```

```

*   endm

```

```

.....
*   macro

```

```

*   buf_rinit    &ptr, &bno, &buf

```

```

*   clr.b        &bno                 ; bno=0
*   move.l       &buf, &ptr           ; ptr=buf

```

```

*   endm

```

```

.....
*   macro

```

```

*   buf_set      &data, &mask

```

```

*   cs.l         &mask, &data         ; data |= mask

```

```

*   endm

```

```

.....
*   macro

```

```

*   buf_get      &data, &bno

```

```

*   subq.b       #1, &bno
*   bts          &bno, &data

```

```

*   endm

```

```

.....
*   macro

```

```

*   buf_winc     &ptr, &data, &mask

```

```

*   lsr.l        #1, &mask            ; mask>>=1
*   bne.s        @cont                ; if non-zero continue
*   move.l       &data, (&ptr)+      ; *ptr++=data
*   clr.l        &data                ; data=0
*   move.l       $80000000, &mask    ; mask=100...

```

- 777 -

Engineering:KlicsCode:CompPict:Bits3.a

?cont

endm

```
-----
macro
buf_rinc    &ptr,&data,&bno
```

```
    cmpi.b    *16,&bno
```

```
    bge.s    @cont
```

```
    swap     &data
```

```
    move.w    (&ptr)+,&data
```

```
    add.b     *16,&bno
```

```
    ; data=*ptr++
```

```
    ; bno-=16
```

@cont

endm

```
-----
macro
buf_flush   &ptr,&data,&mask
```

```
    cmp.l     *$80000000,&mask
```

```
    beq.s     @cont
```

```
    move.l     &data,(&ptr)+
```

```
    ; mask=8000000?
```

```
    ; if buffer empty continue
```

```
    ; *ptr++=data
```

```
endm
-----
```

- 778 -

Engineering: KlicsCode: CompPict: Backward.c

```

.....
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
*...../
/*
Extra fast Backward\convolver
New wavelet coeffs : 3 5 1 1, 1 2 1, 1 1

Optimized for speed:
    dirn = False
    src/dst octave == 0
*/

#define BwdS0(addr0,dAG,dAH,dBH) \
    v=(short *)addr0; \
    dAG= -v; \
    dAH= v; \
    dBH= v<<1; \

#define BwdS1(addr1,addr0,dAG,dAH,dBH) \
    v=(short *)addr1; \
    dBH+= v>>1; \
    dAG+= v-(vs=v<<1); \
    dAH+= v-(vs<<=1); \
    *(short *)addr0=dBH>>1; \

#define Bwd2(addr2,dAG,dAH,dBG,dBH) \
    v=(short *)addr2; \
    dBG= -v; \
    dBH= v; \
    dAH+= v-(vs=v<<1); \
    dAG+= v-(vs<<=1); \

#define Bwd3(addr3,addr2,addr1,dAG,dAH,dBG,dBH) \
    v=(short *)addr3; \
    dAH+= v; \
    dAG+= v; \
    dBG+= v-(vs=v<<1); \
    dBH+= v-(vs<<=1); \
    *(short *)addr1=(dAH-1)>>2; \
    *(short *)addr2=(dAG-1)>>2; \

#define Bwd0(addr0,dAG,dAH,dBG,dBH) \
    v=(short *)addr0; \
    dAG= -v; \
    dAH= v; \
    dBH+= v-(vs=v<<1); \
    dBG+= v-(vs<<=1); \

#define Bwd1(addr1,addr0,addr3,dAG,dAH,dBG,dBH) \
    v=(short *)addr1; \
    dBH+= v; \
    dBG+= v; \
    dAG+= v-(vs=v<<1); \
    dAH+= v-(vs<<=1); \
    *(short *)addr3=(dBH-1)>>2; \
    *(short *)addr0=(dBG-1)>>2; \

#define BwdE2(addr2,dAG,dAH,dBH) \

```

- 779 -

Engineering: XlincsCode: CompPict: Backward.c

```

v=*(short *)addr2; \
dBH= vs=v<<1; \
DAH= v-(vs=v<<1); \
dAG= v+(vs=v<<1);

#define BwdE3(addr3,addr2,addr1,dAG,DAH,dBH) \
v=*(short *)addr3; \
DAH+= v; \
dAG+= v; \
dBH+= v+(vs=v<<1); \
dBH+= v+(vs=v<<1); \
*(short *)addr1=(DAH+1)>>2; \
*(short *)addr2=(dAG-1)>>2; \
*(short *)addr3=dBH>>1;

#define Bwd(base,end,inc) \
addr0=base; \
addr3=addr0-(inc>>2); \
addr2=addr3-(inc>>2); \
addr1=addr2-(inc>>2); \
BwdS0(addr0,dAG,DAH,dBH); \
addr1+=inc; \
BwdS1(addr1,addr0,dAG,DAH,dBH); \
addr2+=inc; \
while(addr2<end) { \
    Bwd2(addr2,dAG,DAH,dBG,dBH); \
    addr3+=inc; \
    Bwd3(addr3,addr2,addr1,dAG,DAH,dBG,dBH); \
    addr0+=inc; \
    Bwd0(addr0,dAG,DAH,dBG,dBH); \
    addr1+=inc; \
    Bwd1(addr1,addr0,addr3,dAG,DAH,dBG,dBH); \
    addr2+=inc; \
} \
BwdE2(addr2,dAG,DAH,dBH); \
addr3+=inc; \
BwdE3(addr3,addr2,addr1,dAG,DAH,dBH);

#define BwdS0r2(addr0,dAG,DAH,dBH) \
v=*(short *)addr0; \
dAG= 0; \
DAH= v; \
dBH= v; \

#define BwdS1r2(addr1,addr0,dAG,DAH,dBH) \
v=*(short *)addr1; \
dBH+= v>>2; \
dAG+= v; \
DAH+= v<<1; \
*(short *)addr0=dBH;

#define Bwd2r2(addr2,dAG,DAH,dBG,dBH) \
v=*(short *)addr2; \
dBG= 0; \
dBH= v; \
DAH+= v; \
dAG+= v<<1;

#define Bwd3r2(addr3,addr2,addr1,dAG,DAH,dBG,dBH) \
v=*(short *)addr3; \
DAH+= 0; \
dAG+= v; \
dBG+= v; \

```

- 780 -

Engineering:KillsCode:CompFicc:Backward.c

```

    DBH-= v<<1; \
    *(short *)addr1=dAH>>1; \
    *(short *)addr2=dAG>>1;

#define Bwd0r2(addr0,dAG,dAH,dBG,DBH) \
    v=(short *)addr0; \
    dAG= 0; \
    dAH= v; \
    DBH-= v; \
    DBG-= v<<1;

#define Bwd1r2(addr1,addr0,addr3,dAG,dAH,dBG,DBH) \
    v=(short *)addr1; \
    DBH-= 0; \
    DBG-= v; \
    dAG-= v; \
    dAH-= v<<1; \
    *(short *)addr3=dBH>>1; \
    *(short *)addr0=dBG>>1;

#define BwdE2r2(addr2,dAG,dAH,DBH) \
    v=(short *)addr2; \
    DBH= v; \
    dAH+= v; \
    dAG+= v<<1;

#define BwdE3r2(addr3,addr2,addr1,dAG,dAH,DBH) \
    v=(short *)addr3; \
    dAH+= 0; \
    dAG+= v; \
    DBH-= v; \
    DBH-= v<<1; \
    *(short *)addr1=dAH>>1; \
    *(short *)addr2=dAG>>1; \
    *(short *)addr3=dBH;

#define Bwdr2(base,end,inc) \
    addr0=base; \
    addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addr1=addr2-(inc>>2); \
    BwdS0r2(addr0,dAG,dAH,DBH); \
    addr1+=inc; \
    BwdS1r2(addr1,addr0,dAG,dAH,DBH); \
    addr2+=inc; \
    while(addr2<end) { \
        Bwd2r2(addr2,dAG,dAH,dBG,DBH); \
        addr3+=inc; \
        Bwd3r2(addr3,addr2,addr1,dAG,dAH,dBG,DBH); \
        addr0+=inc; \
        Bwd0r2(addr0,dAG,dAH,dBG,DBH); \
        addr1+=inc; \
        Bwd1r2(addr1,addr0,addr3,dAG,dAH,dBG,DBH); \
        addr2+=inc; \
    } \
    BwdE2r2(addr2,dAG,dAH,DBH); \
    addr3+=inc; \
    BwdE3r2(addr3,addr2,addr1,dAG,dAH,DBH);

#define BwdS0r3(addr0,dAG,dAH,DBH) \
    v=(short *)addr0; \
    dAG= 0; \
    dAH= 0;

```

- 781 -

Engineering:KlicsCode:CompPict:Backward.c

```

    DBH= v>>1; \

#define BwdS1r3(addr1,addr0,dAG,dAH,DBH) \
    v=(short *)addr1; \
    DBH= v>>1; \
    dAG+= v; \
    dAH-= v; \
    *(short *)addr0=DBH<<1;

#define Bwd2r3(addr2,dAG,dAH,DBG,DBH) \
    v=(short *)addr2; \
    DBG= 0; \
    DBH= 0; \
    dAH-= v; \
    dAG+= v;

#define Bwd3r3(addr3,addr2,addr1,dAG,dAH,DBG,DBH) \
    v=(short *)addr3; \
    dAH+= 0; \
    dAG+= 0; \
    DBG+= v; \
    DBH-= v; \
    *(short *)addr1=dAH; \
    *(short *)addr2=dAG;

#define Bwd0r3(addr0,dAG,dAH,DBG,DBH) \
    v=(short *)addr0; \
    dAG= 0; \
    dAH= 0; \
    DBH+= v; \
    DBG+= v;

#define Bwd1r3(addr1,addr0,addr3,dAG,dAH,DBG,DBH) \
    v=(short *)addr1; \
    DBH+= 0; \
    DBG+= 0; \
    dAG+= v; \
    dAH-= v; \
    *(short *)addr3=DBH; \
    *(short *)addr0=DBG;

#define BwdE2r3(addr2,dAG,dAH,DBH) \
    v=(short *)addr2; \
    DBH= v>>1; \
    dAH+= v; \
    dAG+= v;

#define BwdE3r3(addr3,addr2,addr1,dAG,dAH,DBH) \
    v=(short *)addr3; \
    dAH+= 0; \
    dAG+= 0; \
    DBH+= v; \
    DBH-= v; \
    *(short *)addr1=dAH; \
    *(short *)addr2=dAG; \
    *(short *)addr3=DBH<<1;

#define Bwdr3(base,end,inc) \
    addr0=base; \
    addr3=addr0-(inc>>2); \
    addr2=addr3-(inc>>2); \
    addr1=addr2-(inc>>2); \
    BwdS0r3(addr0,dAG,dAH,DBH); \

```

- 782 -

Engineering:KlacsCode:CompFict:Backward.c

```

addr1+=inc; \
BwdS1r3(addr1,addr0,dAG,dAH,dBH); \
addr2+=inc; \
while(addr2<end) { \
    Bwd2r3(addr2,dAG,dAH,dBG,dBH); \
    addr3+=inc; \
    Bwd3r3(addr3,addr2,addr1,dAG,dAH,dBG,dBH); \
    addr0+=inc; \
    Bwd0r3(addr0,dAG,dAH,dBG,dBH); \
    addr1+=inc; \
    Bwd1r3(addr1,addr0,addr3,dAG,dAH,dBG,dBH); \
    addr2+=inc; \
} \
BwdE2r3(addr2,dAG,dAH,dBH); \
addr3+=inc; \
BwdE3r3(addr3,addr2,addr1,dAG,dAH,dBH);

extern void FASTBACKWARD(char *data, long incl, long loop1, long inc2, char *end2)
extern void HAARBACKWARD(char *data, long incl, long loop1, long inc2, long loop2)
extern void HAARTOPBWD(char *data, long height, long width);
/* extern void HAARXTOPBWD(char *data, long area); */

void FasterBackward(char *data, long incl, long end1, long inc2, char *end2)
{
    register short v, vs, vj, dAG, dAH, dBG, dBH, inc;
    register char *addr0, *addr1, *addr2, *addr3, *end;
    char *base;

    inc=incl;
    for(base=data; base<end2; base+=inc2) {
        end=base+end1;
        Bwd(base, end, inc);
    }
}

extern void TOPBWD(char *data, char *dst, long size_1, long size_0);

void TestTopBackward(short *data, int size[2], int oct_src)
{
    int oct, area=size[0]*size[1]<<1;
    short width=size[0]<<1;
    char *top=area+(char *)data, *left=width+(char *)data;

    for(oct=oct_src-1; oct>0; oct--) {
        long cinc=2<<oct, cinc4=cinc<<2,
            rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in c

        FASTBACKWARD((char *)data, rinc4, area-(rinc<<1), cinc, left);
        FASTBACKWARD((char *)data, cinc4, width-(cinc<<1), rinc, top);
    }
    /* FasterBackward((char *)data, size[0]<<3, area-(size[0]<<2), 2, left);
    FasterBackward((char *)data, 8, width-4, size[0]<<1, top); */
    TOPBWD((char *)data, (char *)data, size[0], size[1]);
}

void TestBackward(data, size, oct_src)

short *data;
int size[2], oct_src;

{
    int oct, area=size[0]*size[1]<<1;
    short width=size[0]<<1;
    char *top=area+(char *)data, *left=width+(char *)data;

```

- 783 -

Engineering:KlcsCode:CompPict:Backward.c

```

for(oct=oct_src-1;oct>=0;oct--) {
    long    cinc=2<<oct, cinc4=cinc<<2,
            rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t

    FasterBackward((char *)data,rinc4,area-(rinc<<1),cinc,left);
    FasterBackward((char *)data,cinc4,width-(cinc<<1),rinc,top);
}

void    Backward3511(data,size,oct_src)
short   *data;
int      size[2], oct_src;
{
    int    oct, area=size[0]*size[1]<<1;
    short  width=size[0]<<1;
    char    *top=area+(char *)data, *left=width+(char *)data;

    for(oct=oct_src-1;oct>=0;oct--) {
        long    cinc=2<<oct, cinc4=cinc<<2,
                rinc=size[0]<<oct+1, rinc4=rinc<<2; /* col and row increments in t

        BACK3511((char *)data,rinc4,area-(rinc<<1),cinc,left);
        BACK3511((char *)data,cinc4,width-(cinc<<1),rinc,top);
    }
    BACK3511V((char *)data,size[0]<<3,area-(size[0]<<2).4,left);
    BACK3511H((char *)data,8,width-4,size[0]<<1,top);
    /* TOPBWD((char *)data,(char *)data,size[1],size[0]);*/
}

```


Engineering:KlicsCode:CompPict:Backward.a

```

-----
*
* Copyright 1993 KLICS Limited
* All rights reserved.
*
* Written by: Adrian Lewis
*
-----
*
* 680X0 3511 Backward code
*
* Coeffs 11 19 5 3
* become 3 5 1 1
*
-----
*
*      seg      'klics'
*
*      macro
*      BwdStart0      &addr0,&dAG,&dAH,&dBH
*
*      move.w      (&addr0),&dAH      ; dAH=*(short *)&addr0
*      move.w      &dAH,&dAG          ; dAG=v
*      neg.w       &dAG              ; dAG=-dAG
*      move.w      &dAH,&dBH          ; dBH=v
*      add.w       &dBH,&dBH          ; dBH=v<<1
*
*      endm
*
*      macro
*      BwdStart1      &addr1,&addr0,&dAG,&dAH,&dBH
*
*      move.w      (&addr1),d0        ; v=*(short *)&addr1
*      move.w      d0,d1              ; vs=v
*      asr.w       #1,d1              ; vs=v>>1
*      add.w       d1,&dBH            ; dBH+=v>>1
*      add.w       d0,&dAG            ; dAG+=v
*      sub.w       d0,&dAH            ; dAH-=v
*      add.w       d0,d0              ; v<<=1
*      add.w       d0,&dAG            ; dAG+=2v
*      add.w       d0,d0              ; v<<=1
*      sub.w       d0,&dAH            ; dAH-=4v
*      asr.w       #1,&dBH            ; dBH>>=1
*      move.w      &dBH,(&addr0)      ; *(short *)&addr0=dBH
*
*      endm
*
*      macro
*      BwdEven &addr2,&dAG,&dAH,&DBG,&dBH
*
*      move.w      (&addr2),d0        ; v=*(short *)&addr2
*      move.w      d0,&dBH            ; dBH=v
*      move.w      d0,&DBG            ; DBG=v
*      neg.w       &DBG              ; DBG=-v
*      add.w       d0,&dAH            ; dAH+=v
*      add.w       d0,&dAG            ; dAG+=v
*      add.w       d0,d0              ; 2v
*      add.w       d0,&dAH            ; dAH+=v
*      add.w       d0,d0              ; 2v
*      add.w       d0,&dAG            ; dAH+=v
*
*      endm
*
*      macro

```

- 785 -

Engineering:KlacsCode:CompPict:Backward.a

```

BwdOdd      &addr3,&addr2,&addr1,&dAG,&dAH,&DBG,&dBH

```

```

move.w      (&addr3),d0      ; v=*(short *)addr3

add.w        d0,&dAH           ; dAH+=v
add.w        d0,&dAG           ; dAG+=v
add.w        d0,&DBG           ; DBG+=v
sub.w        d0,&dBH           ; dBH-=v
add.w        d0,d0             ; 2v
add.w        d0,&DBG           ; DBG+=v
add.w        d0,d0             ; 4v
sub.w        d0,&dBH           ; dBH-=4v

asr.w        #2,&dAH           ; dAH>>=2
move.w       &dAH,(&addr1)     ; *(short *)addr1=dAH
asr.w        #2,&dAG           ; dAG>>=2
move.w       &dAG,(&addr2)     ; *(short *)addr2=dAG

```

```

endm

```

```

macro
BwdEnd2      &addr2,&dAG,&dAH,&dBH

```

```

move.w      (&addr2),d0      ; v=*(short *)addr2

add.w        d0,&dAH           ; dAH+=v
add.w        d0,&dAG           ; dAG+=v
add.w        d0,d0             ; 2v
move.w       d0,&dBH           ; dBH=2v
add.w        d0,&dAH           ; dAH+=2v
add.w        d0,d0             ; 4v
add.w        d0,&dAG           ; dAG+=4v

```

```

endm

```

```

macro
BwdEnd3      &addr3,&addr2,&addr1,&dAG,&dAH,&dBH

```

```

move.w      (&addr3),d0      ; v=*(short *)addr3
add.w        d0,&dAH           ; dAH+=v
add.w        d0,&dAG           ; dAG+=v
lsl.w        #3,d0            ; 8v
sub.w        d0,&dBH           ; dBH-=8v
asr.w        #2,&dAH           ; dAH>>=2
move.w       &dAH,(&addr1)     ; *(short *)addr1=dAH
asr.w        #2,&dAG           ; dAG>>=2
move.w       &dAG,(&addr2)     ; *(short *)addr2=dAG
asr.w        #1,&dBH           ; dBH>>=1
move.w       &dBH,(&addr3)     ; *(short *)addr3=dBH

```

```

endm

```

```

macro
Bwd          &base,&end,&inc

```

```

movea.l      &base,a0          ; addr0=base
move.l       &inc,d0           ; d0=inc
asr.l        #2,d0             ; d0=inc>>2
movea.l      a0,a3             ; addr3=addr0
suba.l       d0,a3             ; addr3-=*(inc>>2)
movea.l      a3,a2             ; addr2=addr3
suba.l       d0,a2             ; addr2-=*(inc>>2)
movea.l      a2,a1             ; addr1=addr2

```

- 786 -

Engineering:KlicsCode:CompPict:Backward.a

```

suba.l    d0,a1                : addr1+=(inc>>2)
BwdStartC a0,d4,d5,d7          : BwdStart0(addrC,dAG,dAH,dBH)
adda.l    6inc,a1              : addr1+=inc
BwdStart1 a1,a0,d4,d5,d7      : BwdStart1(addr1,addr0,dAG,dAH,dBH)
adda.l    6inc,a2              : addr2+=inc
?do       BwdEven a2,d4,d5,d6,d7 : BwdEven(addr2,dAG,dAH,dBG,dBH)
adda.l    6inc,a3              : addr3+=inc
BwdOdd    a3,a2,a1,d4,d5,d6,d7 : BwdOdd(addr3,addr2,addr1,dAG,dAH,dBG)
adda.l    6inc,a0              : addr0+=inc
BwdEven   a0,d6,d7,d4,d5       : BwdEven(addr0,dBG,dBH,dAG,dAH)
adda.l    6inc,a1              : addr1+=inc
BwdOdd    a1,a0,a3,d6,d7,d4,d5 : BwdOdd(addr1,addr0,addr3,dBG,dBH,dAG)
adda.l    6inc,a2              : addr2+=inc
cmpa.l    a2,6end              : addr2<end
bgt.s     0do                  : while
BwdEnd2   a2,d4,d5,d7          : BwdEnd2(addr2,dAG,dAH,dBH)
adda.l    6inc,a3              : addr3+=inc
BwdEnd3   a3,a2,a1,d4,d5,d7    : BwdEnd3(addr3,addr2,addr1,dAG,dAH,dB

```

endm

Back2511 FUNC EXPORT

```

PS      RECORD      8
data    DS.L         1
inci    DS.L         1
endl    DS.L         1
inc2     DS.L         1
end2     DS.L         1
        ENDR

```

```

link     a6,#0          : no local variables
movem.l  d4-d7/a3-a5,-(a7) : store registers

        move.l    PS.incl(a6),d3      : inc=incl
        movea.l   PS.data(a6),a5      : base=data
?do      movea.l   a5,a4               : end=base
        adda.l    PS.end1(a6),a4      : end+=end1
        Bwd       a5,a4,d3            : Bwd(base,end,inc)
        adda.l    PS.inc2(a6),a5      : base+=inc2
        cmpa.l    PS.end2(a6),a5      : end2>base
        blt.w     0do                 : for

        movem.l   (a7)+,d4-d7/a3-a5    : restore registers
        unlk      a6                  : remove locals
        rts                          : return

```

ENDFUNC

```

macro
BwdStartV0 6addr0,6dAG,6dAH,6dBH

        move.l    (6addr0),6dAH      : dAH=(short *)addr0
        move.l    6dAH,6dAG          : dAG=v
        neg.l     6dAG               : dAG=-dAG
        move.l    6dAH,6dBH          : dBH=v
        add.l     6dBH,6dBH          : dBH=v<<1

```

endm

```

macro
BwdStartV1 6addr1,6addr0,6dAG,6dAH,6dBH

```

- 787 -

Engineering:KlicsCode:CompPict:Backward.a

```

move.l    (&addr1),d0      ; v=*(short *)addr1
move.l    d0,d1           ; v=v
asr.l     #1,d1           ; v=v>>1
add.l     d1,&dBH          ; dBH+=v>>1
add.l     d0,&dAG          ; dAG+=v
sub.l     d0,&dAH          ; dAH-=v
add.l     d0,d0           ; v<<=1
add.l     d0,&dAG          ; dAG+=2v
add.l     d0,d0           ; v<<=1
sub.l     d0,&dAH          ; dAH-=4v

asr.l     #1,&dBH          ; dBH>>=1
add.w     &dBH,&dBH        ; shift word back
asr.w     #1,&dBH          ; dBH>>=1
move.l    &dBH,(&addr0)   ; *(short *)addr0=dBH

```

endm

```

macro
BwdEvenV    &addr2,&dAG,&dAH,&DBG,&dBH

```

```

move.l    (&addr2),d0      ; v=*(short *)addr2
move.l    d0,&dBH          ; dBH=v
move.l    d0,&DBG          ; DBG=v
neg.l     &DBG             ; DBG=-v
add.l     d0,&dAH          ; dAH+=v
add.l     d0,&dAG          ; dAG+=v
add.l     d0,d0           ; 2v
add.l     d0,&dAH          ; dAH+=2v
add.l     d0,d0           ; 2v
add.l     d0,&dAG          ; dAG+=2v

```

endm

```

macro
BwdOddV     &addr3,&addr2,&addr1,&dAG,&dAH,&DBG,&dBH

```

```

move.l    (&addr3),d0      ; v=*(short *)addr3

add.l     d0,&dAH          ; dAH+=v
add.l     d0,&dAG          ; dAG+=v
add.l     d0,&DBG          ; DBG+=v
sub.l     d0,&dBH          ; dBH-=v
add.l     d0,d0           ; 2v
add.l     d0,&DBG          ; DBG+=2v
add.l     d0,d0           ; 4v
sub.l     d0,&dBH          ; dBH-=4v

asr.l     #2,&dAH          ; dAH>>=2
lsl.w     #2,&dAH          ; shift word back
asr.w     #2,&dAH          ; dAH>>=2
move.l    &dAH,(&addr1)    ; *(short *)addr1=dAH
asr.l     #2,&dAG          ; dAG>>=2
lsl.w     #2,&dAG          ; shift word back
asr.w     #2,&dAG          ; dAG>>=2
move.l    &dAG,(&addr2)    ; *(short *)addr2=dAG

```

endm

```

macro
BwdEndV2    &addr2,&dAG,&dAH,&dBH

```

```

move.l    (&addr2),d0      ; v=*(short *)addr2

```

- 788 -

Engineering:KlincsCode:CompPict:Backward.a

```

add.l    d0,&dAH      : dAH+=v
add.l    d0,&dAG      : dAG+=v
add.l    d0,d0        : 2v
move.l   d0,&dBH      : dBH=2v
add.l    d0,&dAH      : dAH+=2v
add.l    d0,d0        : 4v
add.l    d0,&dAG      : dAG+=4v

```

```

endm
-----

```

```

macro
BwdEndV3    &addr3,&addr2,&addr1,&dAG,&dAH,&dBH

move.l      (&addr3),d0      : v=(short *)addr3
add.l       d0,&dAH          : dAH+=v
add.l       d0,&dAG          : dAG+=v
lsl.l       #3,d0           : 8v
sub.l       d0,&dBH          : dBH-=8v
asr.l       #2,&dAH          : dAH>>=2
lsl.w       #2,&dAH          : shift word back
asr.w       #2,&dAH          : dAH>>=2
move.l      &dAH,(&addr1)    : *(short *)addr1=dAH
asr.l       #2,&dAG          : dAG>>=2
lsl.w       #2,&dAG          : shift word back
asr.w       #2,&dAG          : dAG>>=2
move.l      &dAG,(&addr2)    : *(short *)addr2=dAG
asr.l       #1,&dBH          : dBH>>=1
lsl.w       #1,&dBH          : shift word back
asr.w       #1,&dBH          : dBH>>=2
add.l       &dBH,&dBH        : dBH<<=1
move.l      &dBH,(&addr3)   : *(short *)addr3=dBH

```

```

endm
-----

```

```

macro
BwdV        &base,&end,&inc

```

```

movea.l    &base,a0          : addr0=base
move.l     &inc,d0           : d0=inc
asr.l      #2,d0             : d0=inc>>2
movea.l    a0,a3            : addr3=addr0
suba.l     d0,a3             : addr3-=inc>>2
movea.l    a3,a2            : addr2=addr3
suba.l     d0,a2            : addr2-=inc>>2
movea.l    a2,a1            : addr1=addr2
suba.l     d0,a1            : addr1-=inc>>2
BwdStartV0 a0,d4,d5,d7      : BwdStart0(addr0,dAG,dAH,dBH)
adda.l     &inc,a1           : addr1+=inc
BwdStartV1 a1,a0,d4,d5,d7   : BwdStart1(addr1,addr0,dAG,dAH,dBH)
adda.l     &inc,a2           : addr2+=inc
@do
BwdEvenV   a2,d4,d5,d6,d7   : BwdEven(addr2,dAG,dAH,dBG,dBH)
adda.l     &inc,a3           : addr3+=inc
BwdOddV    a3,a2,a1,d4,d5,d6,d7 : BwdOdd(addr3,addr2,addr1,dAG,dAH,dBG)
adda.l     &inc,a0           : addr0+=inc
BwdEvenV   a0,d6,d7,d4,d5   : BwdEven(addr0,dBG,dBH,dAG,dAH)
adda.l     &inc,a1           : addr1+=inc
BwdOddV    a1,a0,a3,d6,d7,d4,d5 : BwdOdd(addr1,addr0,addr3,dBG,dBH,dAG)
adda.l     &inc,a2           : addr2+=inc
cmpa.l     a2,&end           : addr2<end
bgt.s      @do              : while
BwdEndV2   a2,d4,d5,d7      : BwdEnd2(addr2,dAG,dAH,dBH)
adda.l     &inc,a3          : addr3+=inc

```

- 789 -

Engineering:KlicsCode:CompPict:Backward.a

```

BwdEndV3      a3,a2,a1,d4,d5,d7      : BwdEnd3(addr3,addr2,addr1,dAG,dAH,dB
.
endm
-----
BackJ511V     FUNC      EXPORT
.
PS      RECORD      6
data    DS.L        1
incl    DS.L        1
end1    DS.L        1
inc2    DS.L        1
end2    DS.L        1
        ENDR
.
        link        a6,#0              ; no local variables
        movem.l     c4-d7/a3-a5,-(a7)  ; store registers
.
        move.l      PS.incl(a6),d3      ; inc=incl
        movea.l     PS.data(a6),a5      ; base=data
3do     movea.l     a5,a4                ; end=base
        adda.l      PS.end1(a6),a4      ; end=end1
        BwdV        a5,a4,d3            ; Bwd(base,end,inc)
        adda.l      PS.inc2(a6),a5      ; base+=inc2
        cmpa.l      PS.end2(a6),a5      ; end2>base
        blt.w       0do                 ; for
.
        movem.l     (a7)+,d4-d7/a3-a5   ; restore registers
        unlk        a6                  ; remove locals
        rts          ; return
.
ENDFUNC
-----
macro
BwdStartH     &addrR,&A,&C
.
        move.l      (&addrR)+,&A      ; 1H1G=(long *)addrR
        move.l      &A,d0              ; A=1H1G, d0=1H1G
        move.l      &A,&C              ; A=1H1G, d0=1H1G, C=1H1G
        add.w       &A,d0              ; A=1H1G, d0=1H2G, C=1H1G
        add.w       d0,&A              ; A=1H3G, d0=1H2G, C=1H1G
        add.w       &A,d0              ; A=1H3G, d0=1H3G, C=1H1G
        swap        &A                ; A=3GH1, d0=1H5G, C=1H1G
        sub.l       d0,&A              ; A=AAAA, d0=1H5G, C=1H1G
.
        endm
-----
macro
BwdCycleH     &addrR,&addrW,&A,&B,&C
.
        move.l      (&addrR)+,&B      ; 1H1G=(long *)addrR
        move.l      &B,d0              ; B=1H1G, d0=1H1G
        add.l       d0,d0              ; B=1H1G, d0=2H2G
        move.l      d0,d1              ; B=1H1G, d0=2H2G, d1=2H2G
        add.l       &B,d0              ; B=1H1G, d0=3H3G, d1=2H2G
        add.l       d0,d1              ; B=1H1G, d0=3H3G, d1=5H5G
        move.l      &B,d2              ; B=1H1G, d0=3H3G, d1=5H5G, d2=1H1G
        move.w      d1,d2              ; B=1H1G, d0=3H3G, d1=5H5G, d2=1H5G
        move.w      &B,d1              ; B=1H1G, d0=3H3G, d1=5H1G, d2=1H5G
        move.w      d0,&B              ; B=1H3G, d0=3H3G, d1=5H1G, d2=1H5G
        move.w      d1,d0              ; B=1H3G, d0=3H1G, d1=5H1G, d2=1H5G
        swap        &B                ; B=3G1H, d0=3H1G, d1=5H1G, d2=1H5G
        swap        d0                ; B=3G1H, d0=1G3H, d1=5H1G, d2=1H5G
.

```

- 790 -

Engineering:KlidesCode:CompPict:Backward.a

```

sub.l    d2,&B      : B=3G1H-1H3G
add.l    d0,&A      : A+=1H3G
add.l    d1,&A      : A+=5G1H

asr.w    #2,&A      : A0>>=2
move.w   &A,&C      : C complete
asr.l    #2,&A      : A1>>=2
move.l   &C,(&addrW)+ : *(long *)addrW=DD
move.l   &A,&C      : C=A1XX

```

endm

```

macro
BwdEndH    &addrR,&addrW,&A,&B,&C

```

```

move.l    (&addrR)+,d0 : 1H1G=*(long *)addrR
move.w    d0,d2        : d2=1G
lsl.w     #2,d2        : d2=4G
neg.w     d2           : d2=-4G
swap      d0           : d0=1G1H
add.w     d0,d2        : d2+=1H
move.l    d0,d1        : d0=1G1H, d1=1G1H
add.w     d0,d1        : d0=1G1H, d1=1G2H
add.w     d1,d0        : d0=1G3H, d1=1G2H
add.w     d0,d1        : d0=1G3H, d1=1G3H
swap      d1           : d0=1G3H, d1=5H1G
add.l     d0,&A        : A+=1G3H
add.l     d1,&A        : A+=5H1G

asr.w     #2,&A        : A1>>=2
move.w    &A,&C        : C complete
asr.l     #2,&A        : A0>>=2
move.l    &C,(&addrW)+ : *(long *)addrW=C
move.w    d2,&A        : A=D1D2
move.l    &A,(&addrW)+ : *(long *)addrW=A

```

endm

```

macro
BwdH        &base,&end,&inc

```

```

movea.l    &base,a0          : addrR=base
movea.l    a0,a1            : addrW=addrR
BwdStartH  a0,d3,d5          : BwdStart(addrR,A,DD)
BwdCycleH  a0,a1,d3,d4,d5    : BwdCycle(addrR,addrW,A,B,C)
BwdCycleH  a0,a1,d4,d3,d5    : BwdCycle(addrR,addrW,B,A,C)
cmpa.l     a0,&end           : addr2<end
bgt.s      %do              : while
BwdEndH    a0,a1,d3,d4,d5    : BwdEnd(addrR,addrW,A,B,DD)

```

endm

Back3511H FUNC EXPORT

```

PS      RECORD      8
data    DS.L        1
incl    DS.L        1
endl    DS.L        1
inc2    DS.L        1
end2    DS.L        1
        ENDR

```

```

link     a6,a0          : no local variables

```

- 791 -

Engineering:Kl:csCode:CompPic:Backward.a

Page 3

```

movem.l    d4-d7/a3-a5, -(a7)    ; store registers
.
move.l     PS.inc1(a6), d3        ; inc=inc1
movea.l    PS.data(a6), a5        ; base=data
edo        a5, a4                ; end=base
adda.l     PS.end1(a6), a4        ; end+=end1
bwdH       a5, a4, d3             ; Bwd(base, end, inc)
adda.l     PS.inc2(a5), a5        ; base+=inc2
cmpa.l     PS.end2(a6), a5        ; end2>base
blt.w      edo                   ; for
.
movem.l     (a7)+, d4-d7/a3-a5    ; restore registers
unlk       a6                    ; remove locals
rts        ; return
.
ENDFUNC
-----
END

```


- 792 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

.....
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
*...../
/*
*  Full still/video Knowles-Lewis Image KlicsEncode System utilising HVS properties
*  and delta-tree coding
*
*  Recoded and re-rationalised (Stand alone version)
*/

#include <PixMath.h>
#include <Bits3.h>
#include <Klics.h>
#include <KlicsHeader.h>
#include <KlicsEncode.h>

#include <Math.h>

/* If bool true the negate value */
#define negif(bool,value) ((bool)?-(value):(value))
#define abs(value) negif(value<0,value)

extern void HaarForward();
extern void DaubeForward();

/* Use the bit level file macros (Bits2.h)
buf_use;*/

/* Huffman encode a block */
#define HuffEncLev(lev,buf) \
    HuffEncode(lev[0],buf); \
    HuffEncode(lev[1],buf); \
    HuffEncode(lev[2],buf); \
    HuffEncode(lev[3],buf);

/* Fixed length encode block of integers */
#define IntEncLev(lev,lpf_bits,buf) \
    IntEncode(lev[0],lpf_bits,buf); \
    IntEncode(lev[1],lpf_bits,buf); \
    IntEncode(lev[2],lpf_bits,buf); \
    IntEncode(lev[3],lpf_bits,buf);

/* Define write a zero */
#define Token0 \
    buf_winc(buf);

/* Define write a one */
#define Token1 \
    buf_set(buf); buf_winc(buf);

/* Write block for data and update memory */
#define DoXfer(addr,pro,lev,dst,mode,oct,nmode,buf) \
    HuffEncLev(lev,buf); \
    PutData(addr,pro,dst); \
    mode[oct]=oct==0?M_STOP:nmode;

/* Function Name: Quantize

```

- 793 -

Engineering:KlitsCode:CompPict:KlitsEnc.c

* Description: H.261 style quantizer
 * Arguments: new, old - image blocks
 pro, lev - returned values
 q - quantizing divisor
 * Returns: lev is all zero, quantized data (pro) & level (lev)

```

aclean Quantize(int new[4], int old[4], int pro[4], int lev[4], short q)
{
    int blk, half_q=(1<<q)-1>>1;
    for(blk=0;blk<4;blk++) {
        int data=new[blk]-old[blk],
            mag_level=abs(data)>>q;

        mag_level=mag_level>135?135:mag_level;
        lev[blk]=negif(data<0,mag_level);
        pro[blk]=old[blk]-negif(data<0,(mag_level<<q)+(mag_level!=0?half_q:0));
    }
    return(pro[0]==0 && pro[1]==0 && pro[2]==0 && pro[3]==0);
}

```

```

void QuantizeLPF(int new[4],int pro[4],int lev[4],short q)
{
    int blk, half_q=(1<<q)-1>>1;
    for(blk=0;blk<4;blk++) {
        int data=new[blk],
            mag_level=abs(data)>>q;

        lev[blk]=negif(data<0,mag_level);
        pro[blk]=(lev[blk]<<q)+half_q;
    }
}

```

/* Function Name: GuessQuantize
 * Description: Estimate threshold quantiser value
 * Arguments: new, old - image blocks
 q - q weighting factor
 * Returns: estimated q_const

```

float GuessQuantize(int new[4],int old[4],float q)
{
    int blk;
    float qt_max=0.0;

    for(blk=0;blk<4;blk++) {
        int i, data=abs(new[blk]-old[blk]);
        float qt;

        for(i=0;data!=0;i++) data>>=1;
        if (i>0) i--;
        qt=((3<<i)-1)>>1/q;

        qt_max=qt_max>qt?qt_max:qt;
    }
    return(qt_max);
}

```

/* Function Name: IntEncode
 * Description: Write a integer to bit file
 * Arguments: lev - integer to write now signed

- 794 -

Engineering:KlitsCode:CompPict:KlitsEnc.c

```

    bits = no of bits
*/

void IntEncode(int lev, int bits, Buf buf)
{
    /* Old version
    int i;

    for(i=bits-1; i>=0; i--) {
        if (lev & (1<<i)) buf_set(buf);
        buf_winc(buf);
    }
    */
    /* New version
    int i, mag=abs(lev);
    Boolean sign=lev<0;

    if (1<<bits-1 <= mag) mag=(1<<bits-1)-1;
    if (sign) buf_set(buf);
    buf_winc(buf);
    for(i=1<<bits-2; i!=0; i>=1) {
        if (mag&i) buf_set(buf);
        buf_winc(buf);
    }
    */
    /* Hardware comparable version: sign mag(lsb->msb) */
    int i, mag=abs(lev);
    Boolean sign=lev<0;

    if (1<<bits-1 <= mag) mag=(1<<bits-1)-1;
    if (sign) buf_set(buf);
    buf_winc(buf);
    for(i=1; i!=1<<bits-1; i<=1) {
        if (mag&i) buf_set(buf);
        buf_winc(buf);
    }
}

/* Function Name: HuffEncodeSA
 * Description: Write a Huffman coded integer to bit file
 * Arguments: lev - integer value
 * Returns: no of bits used
 */

void HuffEncode(int lev, Buf buf)
{
    /* int level=abs(lev);

    if (level>1) buf_set(buf);
    buf_winc(buf);
    if (level>2 || level==1) buf_set(buf);
    buf_winc(buf);
    if (level!=0) {
        if (lev<0) buf_set(buf);
        buf_winc(buf);
        if (level>2) {
            int i;

            for(i=3; i<level; i++) {
                buf_winc(buf);
            }
            buf_set(buf);
            buf_winc(buf);
        }
    }
    */
}

```

- 795 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

    /*
    ** New version */
    int    level=abs(lev), i;

    if (level!=0) buf_set(buf);
    buf_winc(buf);
    if (level!=0) {
        if (lev<0) buf_set(buf);
        buf_winc(buf);
        if (level<8) {
            while (1<level--)
                buf_winc(buf);
            buf_set(buf);
            buf_winc(buf);
        } else {
            for(i=0;i<7;i++)
                buf_winc(buf);
            level-=8;
            for(i=1<=6;i!=0;i>=1) {
                if (level&i) buf_set(buf);
                buf_winc(buf);
            }
        }
    }
}

/* Function Name:  KlicsEChannel
 * Description:    Encode a channel of image
 * Arguments:      src - source channel memory
 *                 dst - destination memory (and old for videos)
 *                 octs, size - octaves of decomposition and image dimensions
 *                 normals - HVS weighted normals
 *                 lpf_bits - no of bits for LPF integer (image coding only)
 */

void KlicsEncY(short *src, short *dst, int octs, int size[2], int thresh[5], int co)
{
    int    oct, mask, x, y, sub, tmp, step=2<<octs, blk[4], mode[4], nz, no, base;
    int    addr[4], new[4], old[4], pro[4], lev[4], zero[s]=0,0,0,0;
    Boolean nzflag, noflag, origin;
    int    bitmask=1<<kle->seqh.precision-kle->frmh.quantizer[0]-1;
    Buf    buf=&kle->buf;

    for(y=0;y<size[1];y+=step)
    for(x=0;x<size[0];x+=step)
    for(sub=0;sub<4;sub++) {
        mode[oct=octs-1]=base_mode;
        if (sub==0) mode[oct=octs-1] != M_LPF;
        mask=2<<oct;
        do {
            GetAddr(addr,x,y,sub,oct,size,mask);
            switch(mode[oct]) {
                case M_VOID:
                    GetData(addr,old,dst);
                    if (BlkZero(old)) mode[oct]=M_STOP;
                    else { DoZero(addr,dst,mode,oct); }
                    break;
                case M_SENDIM_STILL:
                    GetData(addr,new,src);
                    nz=Decide(new); nzflag=nz<=thresh[octs-1];
                    if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer[octs-1]))
                        GetData(addr,old,dst);
            }
        } while (0);
    }
}

```

- 796 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

if (BlkZero(old)) {
    Token0;
    mode{oct}=M_STOP;
} else {
    Token1: Token1;
    DoZero(addr,dst,mode,oct);
}
else {
    Token1: Token0;
    DoXfer(addr,pro,lev,dst,mode,oct,M_SENDIN_STILL,buf);
}
break;
case M_SEND:
    GetData(addr,new,src);
    GetData(addr,old,dst);
    nz=Decide(new); nzflag=nz<=thresh{octs-oct};
    if (BlkZero(old)) {
        if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer{octs-o
            Token0;
            mode{oct}=M_STOP;
        } else {
            Token1: Token0;
            DoXfer(addr,pro,lev,dst,mode,oct,M_SENDIN_STILL,buf);
        }
    } else {
        int oz=Decide(old); no=DecideDelta(new,old);
        Boolean motion=(nz+oz)>>oct <= no; /* motion detection */
        no=DecideDelta(new,old); noflag=no<=compare{octs-oct};
        origin=nz<=no;
        if ((noflag || motion) && !nzflag) { /* was !noflag && !nzfl
            if (Quantize(new,origin?zero:old,pro,lev,kle->frmh.quantizer{o
                Token1: Token1; Token0;
                DoZero(addr,dst,mode,oct);
            } else {
                if (origin) {
                    Token1: Token0;
                    DoXfer(addr,pro,lev,dst,mode,oct,M_SENDIN_STILL,buf);
                } else {
                    Token1: Token1; Token1;
                    DoXfer(addr,pro,lev,dst,mode,oct,M_SEND,buf);
                }
            }
        } else {
            if ((motion || origin) && nzflag) { /* was origin && nzfla
                Token1: Token1; Token0;
                DoZero(addr,dst,mode,oct);
            } else {
                Token0;
                mode{oct}=M_STOP;
            }
        }
    }
    break;
case M_STILL:
    GetData(addr,new,src);
    nz=Decide(new); nzflag=nz<=thresh{octs-oct};
    if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer{octs-oct})
        Token0;
        mode{oct}=M_STOP;
    } else {
        Token1;
        DoXfer(addr,pro,lev,dst,mode,oct,M_STILL,buf);
    }

```

- 797 -

Engineering:KlicsCcdc:CompPict:KlicsEnc.c

```

    }
    break;
case M_LPFIM_STILL:
    GetData(addr,new.src);
    QuantizeLPF(new.pro,lev,kle->frmh.quantizer[0]);
    VerifyData(lev[0],bitmask,tmp);
    VerifyData(lev[1],bitmask,tmp);
    VerifyData(lev[2],bitmask,tmp);
    VerifyData(lev[3],bitmask,tmp);
    IntEncLev(lev,kle->seqh.precision-kle->frmh.quantizer[0].buf);
    PutData(addr,pro,dst);
    mode[oct]=M_QUIT;
    break;
case M_LPFIM_SEND:
    GetData(addr,new.src);
    GetData(addr,old.dst);
    no=DecideDelta(new,old); noflag=no<=compare[octs-oct];
    if (noflag) {
        Token0;
    } else {
        Token1;
        Quantize(new,old,pro,lev,kle->frmh.quantizer[0]);
        HuffEncLev(lev,buf);
        PutData(addr,pro,dst);
    }
    mode[oct]=M_QUIT;
    break;
}
switch(mode[oct]) {
case M_STOP:
    StopCounters(mode,oct,mask,blk,x,y,octs);
    break;
case M_QUIT:
    break;
default:
    DownCounters(mode,oct,mask,blk);
    break;
}
} while (mode[oct]!=M_QUIT);
}

void KlicsEncUV(short *src,short *dst,int octs,int size[2],int thresh[5],int c
{
    int oct,mask,x,y,X,Y,sub,tmp,step=4<<octs,blk[4],mode[4],nz,no
    int addr[4],new[4],old[4],pro[4],lev[4],zero[4]={0,0,0,0};
    Boolean nzflag,noflag,origin;
    int bitmask=-1<<kle->seqh.precision-kle->frmh.quantizer[0]-1;
    Buf buf=&kle->buf;

    for(Y=0;Y<size[1];Y+=step)
    for(X=0;X<size[0];X+=step)
    for(y=Y;y<size[1]&&y<Y+step;y+=step>>1)
    for(x=X;x<size[0]&&x<X+step;x+=step>>1)
    for(sub=0;sub<4;sub++) {
        mode[oct=octs-1]=base_mode;
        if (sub==0) mode[oct=octs-1] != M_LPF;
        mask=2<<oct;
        do {
            GetAddr(addr,x,y,sub,oct,size,mask);
            switch(mode[oct]) {
            case M_VOID:
                GetData(addr,old,dst);

```

- 798 -

Engineering:KlipsCode:CompPict:KlipsEnc.c

```

if (BlkZero(old)) mode{oct}=M_STOP;
else { DoZero(addr,dst,mode,oct); }
break;
case M_SENDIM_STILL:
  GetData(addr,new.src);
  nz=Decide(new); nzflag=nz<=thresh{octs-oct};
  if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer{octs-oct}))
    GetData(addr,old.dst);
    if (BlkZero(old)) {
      Token0;
      mode{oct}=M_STOP;
    } else {
      Token1; Token1;
      DoZero(addr,dst,mode,oct);
    }
  } else {
    Token1; Token0;
    DoXfer(addr,pro,lev,dst,mode,oct,M_SENDIM_STILL,buf);
  }
break;
case M_SEND:
  GetData(addr,new.src);
  GetData(addr,old.dst);
  nz=Decide(new); nzflag=nz<=thresh{octs-oct};
  if (BlkZero(old)) {
    if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer{octs-o
      Token0;
      mode{oct}=M_STOP;
    } else {
      Token1; Token0;
      DoXfer(addr,pro,lev,dst,mode,oct,M_SENDIM_STILL,buf);
    }
  }
  } else {
    int oz=Decide(old), no=DecideDelta(new,old);
    Boolean motion=(nz+oz)>>oct <= no; /* motion detection */
    no=DecideDelta(new,old); noflag=no<=compars{octs-oct};
    origin=nz<=no;
    if ((!noflag || motion) && !nzflag) { /* was !noflag && !nzfl
      if (Quantize(new,origin?zero:old,pro,lev,kle->frmh.quantizer{o
        Token1; Token1; Token0;
        DoZero(addr,dst,mode,oct);
      } else {
        if (origin) {
          Token1; Token0;
          DoXfer(addr,pro,lev,dst,mode,oct,M_SENDIM_STILL,buf);
        } else {
          Token1; Token1; Token1;
          DoXfer(addr,pro,lev,dst,mode,oct,M_SEND,buf);
        }
      }
    }
  }
  } else {
    if ((motion || origin) && nzflag) { /* was origin && nzfla
      Token1; Token1; Token0;
      DoZero(addr,dst,mode,oct);
    } else {
      Token0;
      mode{oct}=M_STOP;
    }
  }
}
break;
case M_STILL:

```

- 799 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

GetData(addr,new,src);
nz=Decide(new); nzflag=nz<=thresh{octs-oct};
if (nzflag || Quantize(new,zero,pro,lev,kle->frmh.quantizer{octs-oct});
    Token0;
    mode{oct}=M_STOP;
} else {
    Token1;
    DoXfer(addr,pro,lev,dst,mode,oct,M_STILL,buf);
}
break;
case M_LPFIM_STILL:
    GetData(addr,new,src);
    QuantizeLPP(new,pro,lev,kle->frmh.quantizer{0});
    VerifyData(lev{0},bitmask,tmp);
    VerifyData(lev{1},bitmask,tmp);
    VerifyData(lev{2},bitmask,tmp);
    VerifyData(lev{3},bitmask,tmp);
    IntEncLev(lev,kle->seqh.precision-kle->frmh.quantizer{0},buf);
    PutData(addr,pro,dst);
    mode{oct}=M_QUIT;
    break;
case M_LPFIM_SEND:
    GetData(addr,new,src);
    GetData(addr,old,dst);
    no=DecideDelta(new,old); noflag=no<=compare{octs-oct};
    if (noflag) {
        Token0;
    } else {
        Token1;
        Quantize(new,old,pro,lev,kle->frmh.quantizer{0});
        HuffEncLev(lev,buf);
        PutData(addr,pro,dst);
    }
    mode{oct}=M_QUIT;
    break;
}
switch(mode{oct}) {
case M_STOP:
    StopCounters(mode,oct,mask,blk,x,y,octs);
    break;
case M_QUIT:
    break;
default:
    DownCounters(mode,oct,mask,blk);
    break;
}
} while (mode{oct}!=M_QUIT);
}

/* index to quant and vice versa */
#define i2q(i) ((float)i*HISTO_DELTA/(float)HISTO
#define q2i(q) Fix2Long(X2Fix(q*(float)HISTO/HISTO_DELTA))

/* Function Name: LookAhead
* Description: Examine base of tree to calculate new quantizer value
* Arguments: src - source channel memory
*            dst - destination memory (and old for videos)
*            ocs, size - octaves of decomposition and image dimensions
*            norms - base HVS weighted normals
* Returns: calculates new quant
*/

```


- 800 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

void LookAhead(short *src, short *dst, float norms[5][5], KlicsE kle)
{
    int x, y, sub, index, size[2] = (kle->seqh.sequence_size[0], kle->seqh.sequence_size[1]);
    thresh[HISTO], quact[HISTO], target;
    int new[4], old[4], addr[4], zero[4] = {0, 0, 0, 0};
    float quant;

    for(index=0; index<HISTO; index++) {
        thresh[index]=0;
        quact[index]=0;
    }
    for(y=0; y<size[1]; y+=2<<octs)
    for(x=0; x<size[0]; x+=2<<octs)
    for(sub=1; sub<4; sub++) {
        float q_thresh;
        int nz, no, oz, blk;
        Boolean ozflag, origin, motion;

        GetAddr(addr, x, y, sub, octs-1, size, 1<<octs);
        GetData(addr, new, src);
        GetData(addr, old, dst);
        nz=Decide(new);
        oz=Decide(old);
        no=DecideDelta(new, old);
        ozflag=kle->encl.intra || BlkZero(old);
        origin=nz<no;
        motion=(nz+oz)>>octs <= no;
        q_thresh=(float)nz/DecideDouble(norms[1][1]);
        if (ozflag || origin) {
            float qt=GuessQuantize(new, zero, norms[1][0]);
            q_thresh=q_thresh<qt?q_thresh:qt;
        } else {
            float qt=GuessQuantize(new, old, norms[1][0]);
            q_thresh=q_thresh<qt?q_thresh:qt;
            if (!motion) {
                qt=(float)no/DecideDouble(norms[1][2]);
                q_thresh=q_thresh<qt?q_thresh:qt;
            }
        }
        index=q2i(q_thresh);
        index=index<0?0:index>HISTO-1?HISTO-1:index;
        thresh[index]++;
    }
    for(index=HISTO-1; index>=0; index--)
        quact[index]=thresh[index]*index+(index==HISTO-1?0:quact[index+1]);

    /* buffer must be greater than bfp_in after this frame */
    /* buffer must be less than buff_size-bfp_in */
    target=kle->encl.bpf_out*kle->encl.prevquact/kle->encl.prevbytes; /* previous
    index=1;
    while(index<HISTO && quact[index]/index>target) index++;
    quant=i2q(index);

    kle->encl.tmp_quant=(kle->encl.tmp_quant+quant)/2.0;
    kle->encl.tmp_quant=i2q((index=q2i(kle->encl.tmp_quant))); /* forward and reve
    kle->encl.prevquact=quact[index]/(index==0?1:index);
}

/* Function Name: BaseNormals

```

- 801 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

* Description:   Calculates base HVS weighted normals
* Arguments:    norms - storage for normals
* Returns:      weighted normals
*/

void BaseNormals(float norms[5][3],KlicsE kle)
{
    float base_norm[3]=(1.0,kle->encl.thresh,kle->encl.compare);
    int norm, oct;

    for(oct=0;oct<5;oct++)
        for(norm=0;norm<3;norm++)
            norms[oct][norm]=base_norm[norm]*(float)(1<<kl)
}

/* Function Name: Normals
* Description:   Calculates HVS weighted normals @ quant
* Arguments:    norms - storage for normals
* Returns:      weighted normals and LPF bits
*/

void Normals(float base_norms[5][3],int thresh[5],int compare[5],KlicsE kle)
{
    int oct, i, norm;

    for(oct=0;oct<=kle->seqh.octaves[0];oct++) {
        norm=Fix2Long(X2Fix(base_norms[oct][0]*kle->encl.tmp_quant));
        norm=norm<171?norm:
        for(i=0;i!=(norm-3);i++)
            norm=norm>>1;
        switch(norm) {
            case 1:
                kle->frmh.quantizer[oct]=i;
                break;
            case 2:
                kle->frmh.quantizer[oct]=i+1;
                break;
            case 3:
            case 4:
                kle->frmh.quantizer[oct]=i+2;
        }
        thresh[oct]=Fix2Long(X2Fix(DecideDouble(base_norms[oct][1]*kle->encl.tmp_q
        compare[oct]=Fix2Long(X2Fix(DecideDouble(base_norms[oct][2]*kle->encl.tmp_
    )
    kle->frmh.quantizer[0]=kle->frmh.quantizer[0]<3?3:kle->frmh.quantizer[0];
    /* minimum 4 bits of quant for lpf due to dynamic range problems */
}

Boolean KlicsFlags(KlicsE kle)
{
    Boolean skip=false;

    kle->encl.buffer=kle->encl.bpf_in;
    kle->frmh.flags=0;
    if (kle->encl.buffer<0)
        kle->encl.buffer=0;
    if (kle->encl.intra)
        kle->frmh.flags |= KFH_INTRA;
    else
        if (skip=kle->encl.buf_sw && kle->encl.buffer>=kle->encl.buf_size)
            kle->frmh.flags |= KFH_SKIP;
    return(skip);
}

```

- 802 -

Engineering:KlicsCode:CompPict:KlicsEnc.c

```

* Function Name: KlicsEncode
* Description:   Encode a frame from YUV (de)transformed image
* Arguments:    src - source image(s)
*               dst - transformed destination memory (and old for videos)

```

```

long KlicsEncode(short *src[3], short *dst[3], KlicsE kle)
{
    float base_norms[5][3];
    int channel, thresh[5], compare[5];
    Buf buf=&kle->buf;

    buf_winit(buf);
    if (KlicsFlags(kle))
        kle->frmh.length=0;
    else {
        for(channel=0; channel<kle->seqh.channels; channel++) {
            int size[2]=(kle->seqh.sequence_size[0]>>(channel==0?0:kle->seqh.s
                kle->seqh.sequence_size[1]>>(channel==0?0:kle->seqh.sub_s
                area=size[0]*size[1], octs=kle->seqh.octaves[channel==0?0:

            switch(kle->seqh.wavelet) {
            case WT_Haar:
                HaarForward(src[channel], size, octs);
                break;
            case WT_Daub4:
                Daub4Forward(src[channel], size, octs);
                break;
            }

            BaseNormals(base_norms, kle);
            if (kle->encd.auto_q && !kle->encd.intra)
                LookAhead(src[0], dst[0], base_norms, kle);
            else
                kle->encd.tmp_quant=kle->encd.quant;
            Normals(base_norms, thresh, compare, kle);
            for(channel=0; channel<kle->seqh.channels; channel++) {
                int size[2]=(kle->seqh.sequence_size[0]>>(channel==0?0:kle->seqh.s
                    kle->seqh.sequence_size[1]>>(channel==0?0:kle->seqh.sub_s
                    octs=kle->seqh.octaves[channel==0?0:1];

                if (kle->encd.intra)
                    KLZERO(dst[channel], size[0]*size[1]);
                if (channel==0) KlicsEncY(src[channel], dst[channel], octs, size, thresh, c
                else KlicsEncUV(src[channel], dst[channel], octs, size, thresh, compare, kle
            }
            buf_flush(buf);
            kle->frmh.length=buf_size(buf);
            kle->encd.buffer+=kle->frmh.length;
            if (!kle->encd.intra)
                kle->encd.prevbytes=kle->frmh.length;
        }
    }
    return(kle->frmh.length);
}

```

- 803 -

Engineering:KlicsCode:CompPict:KlicsHeader.h

```

.....
*
*  © Copyright 1993 KLICS Limited
*  All rights reserved.
*
*  Written by: Adrian Lewis
*
...../
*
*  Sequence and frame headers for Klics-Encoded files
*  High byte first
*/

typedef struct (
    unsigned short  description_length; /* Fixed      - Size of this or parent struc
    unsigned char   version_number[2]; /* Fixed      - Version and revision numbers
) KlicsHeader;

typedef struct (
    KlicsHeader head; /* Fixed      - Size and version of this str
    unsigned short sequence_size[3]; /* Source    - Luminance dimensions and num
    unsigned char  channels; /* Source    - Number of channels: 3 - YUV,
    unsigned char  sub_sample[2]; /* Source    - UV sub-sampling in X and Y d
    unsigned char  wavelet; /* Source    - Wavelet used: 0 - Haar, 1 -
    unsigned char  precision; /* Source    - Bit precision for transform
    unsigned char  octaves[2]; /* Source    - Number of octaves Y/UV (maxi
    unsigned char  reserved[3]; /* Fixed     - Reserved for future use */
) KlicsSeqHeader;

typedef struct (
    KlicsHeader head; /* Fixed      - Size and version of this str
    unsigned long  length; /* Calc      - Length of frame data (bytes)
    unsigned long  frame_number; /* Calc     - Frame number intended for se
    unsigned char  flags; /* Calc      - Bitfield flags: 0 - frame sk
    unsigned char  quantizer[5]; /* Calc     - Quantiser shift values (octav
    unsigned short reserved; /* Fixed     - Reserved for future use */
) KlicsFrameHeader;

#define KFH_SKIP    0x1
#define KFH_INTRA   0x2

/*
*  Implementation notes :
*  QuickTime  Must have KlicsFrameHeader.length set to a valid number
*  Sun       Must have KlicsSeqHeader in data stream
*
*  Possible developments:
*  KlicsFrameHeader.quantizer
*  Currently contains shift rather than step-size
*  Different values for UV and GH, HG, GG sub-bands are not currently suppo
*/

```

- 804 -

Engineering:KlicsCode:Klics Codec:KlicsEncode.r

```

/*
 * KlicsEncode resource file
 */

#include "Types.r"
#include "MPWTypes.r"
#include "ImageCodec.r"

/*
 * Klics Compressor included into the applications resource file here
 */

#define klicsCodecFormatName    "Klics"
#define klicsCodecFormatType    "klic"

/*
 * This structure defines the capabilities of the codec. There will
 * probably be a tool for creating this resource, which measures the performance
 * and capabilities of your codec.
 */
resource 'cdci' (129, "Klics CodecInfo", locked) {
    klicsCodecFormatName,           /* name of the codec TYPE ( da
    1,                             /* version */
    1,                             /* revision */
    'klic',                        /* who made this codec */
    0,
    codecInfoDoes32|codecInfoDoes8|codecInfoDoesTemporal, /* depth and etc suppo
    codecInfoDepth24|codecInfoSequenceSensitive,          /* which data formats do we us
    100,                                                      /* compress accuracy (0-255) (
    100,                                                      /* decompress accuracy (0-255)
    0,                                                         /* millisecs to compress 320x2
    0,                                                         /* millisecs to decompress 320
    0,                                                         /* compression level (0-255) (
    0,
    32,                                                         /* minimum height */
    32,                                                         /* minimum width */
    0,
    0,
    0
};

resource 'thrg' (128, "Klics Compressor", locked) {
    compressorComponentType,
    klicsCodecFormatType,
    'klic',
    codecInfoDoes32|codecInfoDoes8|codecInfoDoesTemporal,
    0,
    'cdci',
    128,
    'STR ',
    128,
    'STR ',
    129,
    'ICON',
    128
};

resource 'STR' (128) {
    "Klics Compress"

```

- 805 -

➤ Engineering:KlacsCode:Klacs Codec:KlacsEncode.r

1:

resource 'STR' (129) {

'Wavelet transform & multiresolution tree based coding scheme'

- 806 -

Engineering:KlicsCode:Klics Codec:KlicsDecode.r

```

/*
 * KlicsDecode resource file
 */

#include "Types.r"
#include "MPWTypes.r"
#include "ImageCodec.r"

/*
 * Klics Compressor included into the applications resource file here
 */

#define klicsCodecFormatName    "Klics"
#define klicsCodecFormatType    "klic"

/*
 * This structure defines the capabilities of the codec. There will
 * probably be a tool for creating this resource, which measures the performance
 * and capabilities of your codec.
 */
resource 'cdci' (129, "Klics CodecInfo", locked) {
    klicsCodecFormatName,           /* name of the codec TYPE ( da
    1,                             /* version */
    1,                             /* revision */
    'klic',                        /* who made this codec */
    codecInfoDoes32|codecInfoDoes16|codecInfoDoes8|codecInfoDoesTemporal|codecInfo
    0,
    codecInfoDepth24|codecInfoSequenceSensitive, /* which data formats do we un-
    100,                             /* compress accuracy (0-255) (
    100,                             /* decompress accuracy (0-255)
    0,                             /* millisecs to compress 320x2
    0,                             /* millisecs to decompress 320
    0,                             /* compression level (0-255) (
    0,
    32,                             /* minimum height */
    32,                             /* minimum width */
    0,
    0,
    0
};

resource 'thng' (130, "Klics Decompressor", locked) {
    decompressorComponentType,
    klicsCodecFormatType,
    'klic',
    codecInfoDoes32|codecInfoDoes16|codecInfoDoes8|codecInfoDoesTemporal|codecInfo
    0,
    'cdec',
    128,
    'STR ',
    130,
    'STR ',
    131,
    'ICON',
    130
};

resource 'STR ' (130) {

```

- 807 -

CLAIMS

WE CLAIM:

1. A method of transforming a sequence of input digital data values into a first sequence of transformed digital data values and of inverse transforming a second sequence of transformed digital data values into a sequence of output digital data values, said sequence of input digital data values comprising a boundary subsequence and a non-boundary subsequence, comprising the steps of:
 - 10 running a number of said input digital data values of said boundary subsequence through a low pass boundary forward transform perfect reconstruction digital filter and through a high pass boundary forward transform perfect reconstruction digital filter to produce a first subsequence of said first sequence of transformed digital data values, said first subsequence of said first sequence of transformed digital data values comprising interleaved low and high frequency transformed digital data values;
 - 20 running a number of said input digital data values of said non-boundary subsequence through a low pass non-boundary forward transform perfect reconstruction digital filter and also through a high pass non-boundary forward transform perfect reconstruction digital filter to produce a second subsequence of said first sequence of transformed digital data values, said second subsequence of said first sequence of transformed digital data values comprising interleaved low and high frequency transformed digital data values, said low pass boundary forward transform perfect reconstruction digital filter having a fewer number of coefficients than said low pass non-boundary forward transform perfect reconstruction digital filter, said high pass boundary forward transform perfect reconstruction digital filter having a fewer number of coefficients

- 808 -

than said high pass non-boundary forward transform perfect reconstruction digital filter;

converting said first sequence of transformed digital data values into said second sequence of transformed digital data values, said second sequence of transformed digital data values comprising a first subsequence of said second sequence of transformed digital data values and a second subsequence of said second sequence of transformed digital data values;

running a number of said first subsequence of said second sequence of transformed digital data values through an interleaved boundary inverse transform perfect reconstruction digital filter to produce at least one output digital data value;

running a number of said second subsequence of said second sequence of transformed digital data values through a first interleaved non-boundary inverse transform perfect reconstruction digital filter to produce output digital data values; and

running a number of said second subsequence of transformed digital data values through a second interleaved non-boundary inverse transform perfect reconstruction digital filter to produce output digital data values, said output digital data values produced by said interleaved boundary inverse transform perfect reconstruction digital filter, said first interleaved non-boundary inverse transform perfect reconstruction digital filter, and said second interleaved non-boundary inverse transform perfect reconstruction digital filter comprising a subsequence of said output digital data values of said sequence of output digital data values.

2. The method of Claim 1, wherein said low pass boundary forward transform perfect reconstruction digital filter has X coefficients and wherein said low pass non-boundary forward transform perfect reconstruction digital

- 809 -

filter has Y coefficients, Y being greater than X , said X coefficients of said low pass boundary forward transform perfect reconstruction digital filter being chosen so that said low pass boundary forward transform perfect reconstruction digital filter outputs a transformed digital data value H_0 when the low pass boundary forward perfect transform reconstruction digital filter operates on input digital data values ID_0-ID_{X-1} adjacent said boundary, said transformed digital data value H_0 being substantially equal to what the output of the low pass non-boundary forward transform perfect reconstruction digital filter would be were the low pass non-boundary forward perfect reconstruction digital filter to operate on ID_0-ID_{X-1} as well as $Y-X$ additional input digital data values outside said boundary, said additional input digital data values having preselected values.

3. The method of Claim 2, wherein $Y-X=1$, wherein there is one additional input digital data value ID_{-1} , and wherein ID_{-1} is preselected to be substantially equal to ID_0 .

4. The method of Claim 2, wherein $Y-X=1$, wherein there is one additional input digital data value ID_{-1} , and wherein ID_{-1} is preselected to be substantially equal to zero.

5. The method of Claim 1, wherein said sequence of input digital data values is a sequence of digital data values associated with pixels of either a row or a column of a two dimensional image, said boundary of said sequence of input digital data values corresponding with either a start or an end of said row or said column.

6. The method of Claim 1, wherein said sequence of input digital data values is a sequence of digital data values associated with an audio signal.

- 810 -

7. The method of Claim 1, wherein said low and high pass non-boundary forward transform perfect reconstruction digital filters are forward transform quasi-perfect reconstruction filters which have coefficients which approximate the coefficients of true forward transform perfect reconstruction filters.

8. The method of Claim 1, wherein said low and high pass non-boundary forward transform perfect reconstruction digital filters are both four coefficient quasi-Daubechies filters the coefficients of which approximate the coefficients of true four coefficient Daubechies filters.

9. The method of Claim 8, wherein one of said four coefficient quasi-Daubechies filters has the coefficients 11/32, 19/32, 5/32 and 3/32 independent of sign.

10. The method of Claim 1, wherein said low pass non-boundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter H of the form:

$$H_n = aID_{2n-1} + bID_{2n} + cID_{2n+1} - dID_{2n+2}$$

n being a positive integer, ID_0-ID_m being input digital data values, m being a positive integer, ID_0 being the first input digital data value in said sequence of input digital data values, and wherein said low pass boundary forward transform perfect reconstruction digital filter is a three coefficient digital filter of the form:

$$H_0 = aID_{-1} + bID_0 + cID_1 - dID_2$$

ID_{-1} being a predetermined input digital data value outside said boundary and having a preselected value.

11. The method of Claim 10, wherein said high pass

- 811 -

non-boundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter of the form:

$$G_n = dID_{2n-1} + cID_{2n} - bID_{2n+1} + aID_{2n+2}$$

5 n being a positive integer, and wherein said high pass boundary forward transform perfect reconstruction digital filter is a three coefficient digital filter of the form:

$$G_0 = dID_{-1} + cID_0 - bID_1 + aID_2$$

dID₋₁ having a preselected value.

10 12. The method of Claim 11, wherein: $a + b + c - d$ is substantially equal to 1, wherein $a - b + c + d$ is substantially equal to 0, and wherein $ac - bd$ is substantially equal to zero.

13. The method of Claim 12, wherein: $a=11/32$,
15 $b=19/32$, $c=5/32$ and $d=3/32$.

14. The method of Claim 11, wherein said interleaved boundary inverse transform perfect reconstruction digital filter is a two coefficient digital filter of the form:

$$OD_0 = 4(b-a)H_0 + 4(c-d)G_0$$

20 wherein OD_0 is an output digital data value of said sequence of output digital data values, wherein G_0 is the output of said high pass boundary forward transform perfect reconstruction digital filter when the high pass boundary forward transform perfect reconstruction digital
25 filter operates on input digital data values ID_0 , ID_1 and ID_2 adjacent said boundary, and wherein H_0 is the output of said low pass boundary forward transform perfect reconstruction digital filter when the low pass boundary

- 812 -

forward transform perfect reconstruction digital filter operates on input digital data values ID_0 , ID_1 and ID_2 adjacent said boundary.

15. The method of Claim 14, wherein one of said first
5 and second interleaved non-boundary inverse transform perfect reconstruction digital filters is of the form:

$$D_{2n+1} = 2(cH_n - bG_n + aH_{n+1} + dG_{n+1})$$

n being a non-negative integer, and wherein the other of
said first and second interleaved non-boundary inverse
10 perfect reconstruction digital filters is of the form:

$$D_{2n+2} = 2(-dH_n + aG_n + bH_{n+1} + cG_{n+1})$$

n being a non-negative integer, wherein H_n , G_n , H_{n+1} and G_{n+1} comprise a subsequence of said second sequence of transformed digital data values.

15 16. The method of Claim 1, wherein said low pass non-boundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter having the coefficients: $11/32$, $19/32$, $5/32$ and $-3/32$, and wherein
20 said high pass non-boundary forward transform perfect reconstruction digital filter is a four coefficient quasi-Daubechies filter having the coefficients: $3/32$, $5/32$, $-19/32$ and $11/32$.

17. The method of Claim 1, wherein said low and high
pass non-boundary forward transform perfect reconstruction
25 digital filters are chosen from the group consisting of:
true six coefficient Daubechies filters and quasi-Daubechies filters, the coefficients of the quasi-Daubechies filters approximating the coefficients of true six coefficient Daubechies filters.

- 813 -

18. The method of Claim 1, further comprising the steps of:

encoding said first sequence of transformed digital data values into an encoded sequence; and
5 decoding said encoded sequence of digital data values into said second sequence of transformed digital data values and supplying said second sequence of transformed digital data values to said interleaved boundary inverse transform perfect reconstruction
10 digital filter, said first interleaved non-boundary inverse transform perfect reconstruction digital filter, and said second interleaved non-boundary inverse transform perfect reconstruction digital filter.

15 19. The method of Claim 18, further comprising the step of:

quantizing each of said digital data values in said first sequence of transformed values before said encoding step.

20 20. The method of Claim 1, wherein each of said input digital data values of said sequence of input digital data values is stored in a separate memory location, and wherein some of said memory locations are overwritten in a sequence with said sequence of transformed digital data values as
25 said digital data input values are transformed into said transformed digital data values.

21. A method of transforming a sequence of input digital data values into a sequence of transformed digital data values, said sequence of input digital data values
30 comprising a boundary subsequence and a non-boundary subsequence, comprising the steps of:

running a number of said input digital data values of said boundary subsequence through a low pass boundary forward transform perfect reconstruction

- 814 -

digital filter and through a high pass boundary
forward transform perfect reconstruction digital
filter to produce a first subsequence of said sequence
of transformed digital data values, said first
5 subsequence of said sequence of transformed digital
data values comprising interleaved low and high
frequency transformed digital data values; and
running a number of said input digital data
values of said non-boundary subsequence through a low
10 pass non-boundary forward transform perfect
reconstruction digital filter and also through a high
pass non-boundary forward transform perfect
reconstruction digital filter to produce a second
subsequence of said sequence of transformed digital
15 data values, said second subsequence of said sequence
of transformed digital data values comprising
interleaved low and high frequency transformed digital
data values, said low pass boundary forward transform
perfect reconstruction digital filter having a fewer
20 number of coefficients than said low pass non-boundary
forward transform perfect reconstruction digital
filter, said high pass boundary forward transform
perfect reconstruction digital filter having a fewer
number of coefficients than said high pass non-
25 boundary forward transform perfect reconstruction
digital filter.

22. A method, comprising the steps of:

generating a sub-band decomposition having a
plurality of octaves, a first of said plurality of
30 octaves comprising at least one first digital data
value, a second of said plurality of octaves
comprising at least one second digital data value;
calculating a sum of the absolute values of said
at least one first digital data value;
35 determining if said at least one first digital
data value is interesting using a first threshold

- 815 -

limit;

calculating a sum of the absolute values of said
at least one second digital data value; and

5 determining if said at least one second digital
data value is interesting using a second threshold
limit.

23. A method of traversing a tree decomposition, said
tree decomposition comprising a plurality of transformed
data values, each of said plurality of transformed data
10 values having a unique address identified by coordinates X
and Y, comprising the step of:

calculating at least four transformed data value
addresses by incrementing a count, the count
comprising one bit $C1_x$ in the X coordinate and one bit
15 $C1_y$ in the Y coordinate, to generate said at least
four transformed data value addresses.

24. A method, comprising the step of:

determining an address of a transformed data value in
a tree decomposition by shifting a value a number of times,
20 said tree decomposition having a number of octaves, said
transformed data value being in one of said octaves, said
number of times being at least dependent upon said one
octave.

25. A method, comprising the step of:

25 determining an address of a transformed data value in
a tree decomposition by multiplying a value by a factor,
said tree decomposition having a number of octaves, said
transformed data value being in one of said octaves, said
factor being at least dependent upon said one octave.

30 26. A method, comprising the step of:

determining an address of a transformed data value in
a tree decomposition by shifting a value a number of times,
said tree decomposition having a number of frequency sub-

- 816 -

bands, said transformed data value being in one of said frequency sub-bands, said number of times being at least dependent upon said frequency sub-band.

27. A method, comprising the step of:

5 determining an address of a transformed data value in a tree decomposition by performing a logical operation upon a value, said tree decomposition having a number of frequency sub-bands, said transformed data value being in one of said frequency sub-bands, said logical operation
10 performed being at least dependent upon said one frequency sub-band.

28. The method of Claim 27, wherein said logical operation is a bit-wise logical AND operation.

29. A method for determining a low pass quasi-perfect
15 reconstruction filter and a high pass quasi-perfect reconstruction filter from a wavelet function, said low pass quasi-perfect reconstruction filter having a plurality of coefficients, said high pass quasi-perfect reconstruction filter having a plurality of coefficients,
20 comprising the steps of:

determining a low pass wavelet digital filter and a high pass wavelet digital filter from said wavelet function, said low pass wavelet digital filter having a plurality of coefficients, said high pass wavelet digital
25 filter having a plurality of coefficients;

choosing the coefficients of said low pass quasi-perfect reconstruction digital filter to be fractions such that when a sequence of data values having values of 1 is processed by said low pass quasi-perfect reconstruction
30 digital filter the output of said low pass quasi-perfect reconstruction digital filter is exactly a power of 2; and

choosing the coefficients of the high pass quasi-perfect reconstruction digital filter to be fractions such that when a sequence of data values having values of 1 is

- 817 -

processed by said high pass quasi-perfect reconstruction digital filter the output of said high pass quasi-perfect reconstruction digital filter is exactly 0, whereby each of the plurality of coefficients of said low pass quasi-
5 perfect reconstruction digital filter is substantially identical to a corresponding one of said plurality of coefficients of said low pass wavelet digital filter, and whereby each of the plurality of coefficients of said high pass quasi-perfect reconstruction digital filter is
10 substantially identical to a corresponding one of said plurality of coefficients of said high pass wavelet digital filter.

30. A method of estimating a compression ratio of a number of original data values to a number of compressed
15 data values at a value of a quality factor Q, comprising the steps of:

examining a first block of transformed data values of a tree, said first block being one of a number of lowest frequency blocks of a high pass component sub-band, said
20 tree being part of a sub-band decomposition; and

determining a value of said quality factor Q at which said data values of said first block would be converted into compressed data values, and not determining a value of said quality factor Q at which any other block of data
25 values of said tree would be converted into a number of compressed data values.

31. The method of Claim 30, wherein said number of original data values represents a frame of an image.

32. The method of Claim 31, further comprising the
30 step of:

determining a number of lowest frequency blocks of said high pass component sub-band which would be converted into compressed data values given a value of said quality factor Q.

- 818 -

33. A method of transforming a sequence of image data values, comprising the step of:

filtering said sequence of image data values using a quasi-perfect reconstruction filter to generate a
5 decomposition having a plurality of octaves, said quasi-perfect reconstruction filter having six coefficients.

34. The method of Claim 33, wherein said six coefficients are selected from the group consisting of:
30/128, 73/128, 41/128, 12/128, 7/128 and 3/128,
10 irrespective of sign.

35. A method of detecting motion in a tree decomposition, said tree decomposition comprising a plurality of octaves of blocks of data values, comprising the steps of:

15 comparing data values of a first block in an octave with data values of a second block in said octave; and generating a token indicating motion based on said comparing.

36. A method, comprising the steps of:

20 generating a sub-band decomposition having a plurality of octaves, a first of said plurality of octaves comprising at least one first digital data value, a second of said plurality of octaves comprising at least one second digital data value;

25 determining if said at least one first digital data value is interesting using a first threshold limit; and determining if said at least one second digital data value is interesting using a second threshold limit.

37. A method, comprising the steps of:

30 generating a sub-band decomposition of a first frame having a plurality of octaves, a first of said plurality of octaves comprising at least one first digital data value, a

- 819 -

second of said plurality of octaves comprising at least one second digital data value;

generating a sub-band decomposition of a second frame having a plurality of octaves, a first of said plurality of 5 octaves comprising at least one first digital data value, a second of said plurality of octaves comprising at least one second digital data value;

comparing said first digital data value of said first frame with said first digital data value of said second 10 frame using a first threshold compare; and

comparing said second digital data value of said first frame with said second digital data value of said second frame using a second threshold compare.

38. A method, comprising the steps of:

15 reading a sequence of data values from a plurality of memory locations, each of said data values being stored in a separate one of said plurality of memory locations; and
overwriting some of said memory locations in a sequence as said data values are transformed into a 20 sequence of transformed data values of a sub-band decomposition.

39. A method, comprising the steps of:

performing a function on a plurality of data values of a new block to generate a first output value, said new 25 block being a block of data values of a sub-band decomposition of a new frame;

performing said function on a plurality of numbers to generate a second output value, each of said numbers substantially equalling a difference of a data value in 30 said plurality of data values of said new block and a corresponding data value in a corresponding plurality of data values of an old block, said old block being a block of data values of a sub-band decomposition of an old frame; and

35 generating a token if said first output value has a

- 820 -

predetermined relationship with respect to said second output value.

40. The method of Claim 39, wherein said token is a SEND_STILL token.

5 41. A method, comprising the steps of:

performing a function on a plurality of data values of a new block to generate a corresponding plurality of output values, said new block being a block of data values of a sub-band decomposition;

10 comparing each of said plurality of output values with a predetermined number; and

generating a token if substantially all of said output values have a predetermined relationship with respect to said predetermined number.

15 42. The method of Claim 41, wherein said token is a VOID token.

43. A method, comprising the steps of:

subtracting each one of a plurality of data values of a new block with a corresponding one of a plurality of data values of a old block to generate a corresponding plurality of output values, said new block being a block of data values of a sub-band decomposition of a new frame, said old block being a block of data values of a sub-band decomposition of a old frame;

25 comparing each of said plurality of output values with a predetermined number; and

generating a token if substantially all of said output values have a predetermined relationship with respect to said predetermined number.

30 44. The method of Claim 43, wherein said token is a VOID token.

- 821 -

45. A method, comprising the steps of:
determining an absolute value for each of a plurality
of data values of a block of a sub-band decomposition;
determining a sum of said absolute values; and
5 generating a token based on a comparison of said sum
with a predetermined number.

46. The method of Claim 45, wherein said token is a
VOID token.

47. A method, comprising the steps of:
10 processing a sequence of first image data values using
a low pass forward transform perfect reconstruction digital
filter and a high pass forward transform perfect
reconstruction digital filter to create a first sequence of
transformed data values, said low pass forward transform
15 perfect reconstruction digital filter and said high pass
forward transform perfect reconstruction digital filter
each having coefficients chosen from a first group of
coefficients independent of sign;

converting said first sequence of transformed data
20 values into a second sequence of transformed data values;
and

using digital circuitry to process said second
sequence of transformed data values using a low pass
inverse transform perfect reconstruction digital filter and
25 a high pass inverse transform perfect reconstruction
digital filter into a sequence of second image data values,
said low pass inverse transform perfect reconstruction
digital filter and said high pass inverse transform perfect
reconstruction digital filter each having coefficients
30 chosen from a second group of coefficients independent of
sign.

48. The method of claim 47, wherein said digital
circuitry used to process said second sequence of
transformed data values is a digital computer having a

- 822 -

microprocessor.

49. The method of claim 47, wherein at least one of the coefficients in said first group of coefficients is not contained in said second group of coefficients.

5 50. The method of claim 47, wherein said first group of coefficients has a different number of coefficients than said second group of coefficients.

51. The method of claim 50, wherein said sequence of first image data values is a sequence of chrominance data 10 values.

52. The method of claim 50, wherein said low pass forward transform perfect reconstruction digital filter and said high pass forward transform perfect reconstruction digital filter each have four coefficients, and wherein 15 said low pass inverse transform perfect reconstruction digital filter and said high pass inverse transform perfect reconstruction digital filter each have two coefficients.

53. The method of claim 52, wherein said sequence of first image data values is a sequence of chrominance data 20 values.

54. The method of claim 47, wherein each of said coefficients of said low pass inverse transform perfect reconstruction digital filter and said high pass inverse transform perfect reconstruction digital filter is selected 25 from the group consisting of: $5/8$, $3/8$ and $1/8$, independent of sign.

55. The method of claim 47, wherein said converting step comprises the steps of:

encoding said first sequence of transformed data 30 values into a compressed data stream; and

- 823 -

decoding said compressed data stream into said second sequence of transformed data values.

56. A method comprising the step of using digital circuitry to process a sequence of image data values using
5 a low pass forward transform perfect reconstruction digital filter and a high pass forward transform perfect reconstruction digital filter to generate a sub-band decomposition, said low pass forward transform perfect reconstruction digital filter and said high pass forward
10 transform perfect reconstruction digital filter each having four coefficients, each of said four coefficients being selected from the group consisting of: $5/8$, $3/8$ and $1/8$, independent of sign.

57. The method of claim 56, wherein said digital
15 circuitry comprises means for low pass forward transform perfect reconstruction digital filtering and for high pass forward transform perfect reconstruction digital filtering.

58. A method comprising the step of using digital circuitry to process a sequence of transformed data values
20 of a sub-band decomposition using an odd inverse transform perfect reconstruction digital filter and an even inverse transform perfect reconstruction digital filter, said odd inverse transform perfect reconstruction digital filter and said even inverse transform perfect reconstruction digital
25 filter each having four coefficients, each of said four coefficients being selected from the group consisting of: $5/8$, $3/8$ and $1/8$, independent of sign.

59. The method of claim 58, wherein said digital circuitry is a digital computer having a microprocessor.

30 60. A method comprising the step of generating a compressed data stream indicative of a video sequence from a sub-band decomposition, said compressed data stream

- 824 -

comprising a first data value, a first token, a second data value, and a second token, said first token being indicative of a first encoding method used to encode said first data value, said second token being indicative of a second encoding method used to encode said second data value, said first token consisting of a first number of bits and said second token consisting of a second number of bits.

61. The method of claim 60, wherein said first encoding method is taken from the group consisting of: SEND mode, STILL_SEND mode, VOID mode, and STOP mode.

62. The method of claim 60, wherein said first token is a single bit token.

63. A method, comprising the steps of:

15 forward transforming image data values to generate a first sequence of transformed data values of a first sub-band decomposition, said first sub-band decomposing having a first number of octaves;

converting said first sequence of transformed data values into a second sequence of transformed data values;

20 using digital circuitry to inverse transforming said second sequence of transformed data values into a third sequence of transformed data values, said third sequence of transformed data values comprising a second sub-band decomposition having a second number of octaves, said second number of octaves being smaller than said first number of octaves, said second sub-band decomposition having a low pass component, said low pass component of said second sub-band decomposition comprising data values

25 indicative of rows of data values of an image, said rows of said image extending in a first dimension, said image also having columns of said data values extending in a second dimension;

expanding said low pass component in said first

30

- 825 -

dimension using interpolation to generate an interpolated low pass component; and

expanding said interpolated low pass component in said second dimension by replicating rows of said data values of 5 said interpolated low pass component.

64. The method of claim 63, wherein said digital circuitry is a digital computer having a microprocessor.

65. The method of claim 63, wherein said converting step comprises the steps of:

10 encoding said first sequence of transformed data values into a compressed data stream comprising tokens and encoded data values; and

decoding said compressed data stream into said second sequence of transformed data values.

1/24

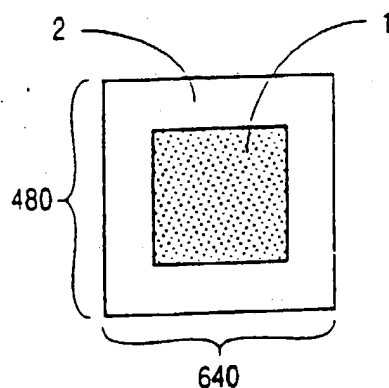


Fig. 1
(PRIOR ART)

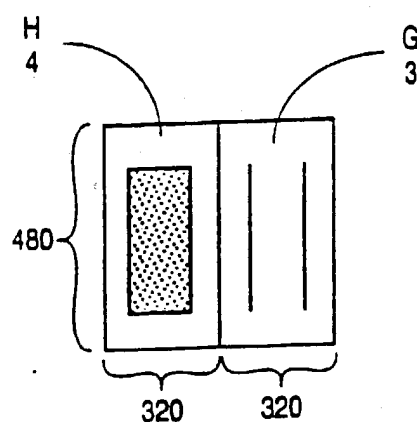


Fig. 2
(PRIOR ART)

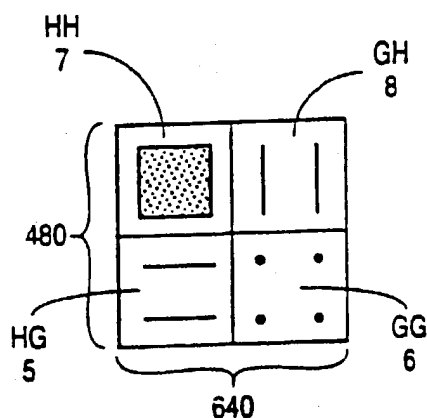


Fig. 3
(PRIOR ART)

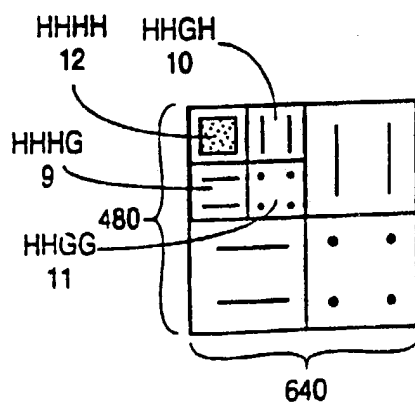


Fig. 4
(PRIOR ART)

2/24

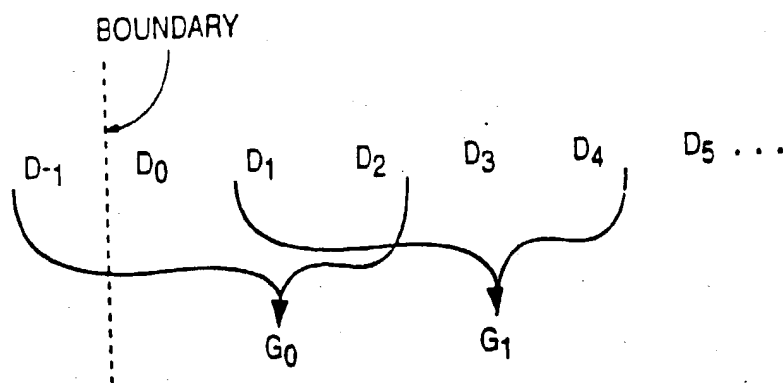


Fig. 5
(PRIOR ART)

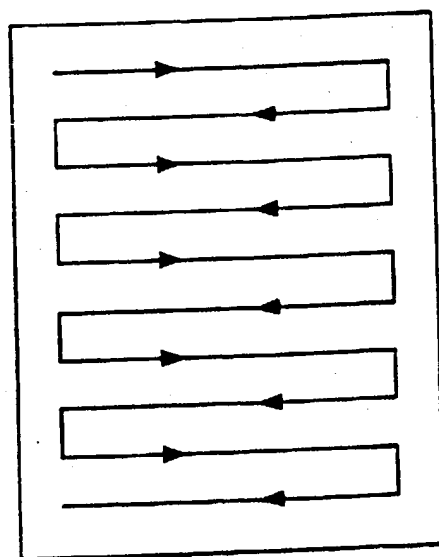


Fig. 6
(PRIOR ART)

3/24

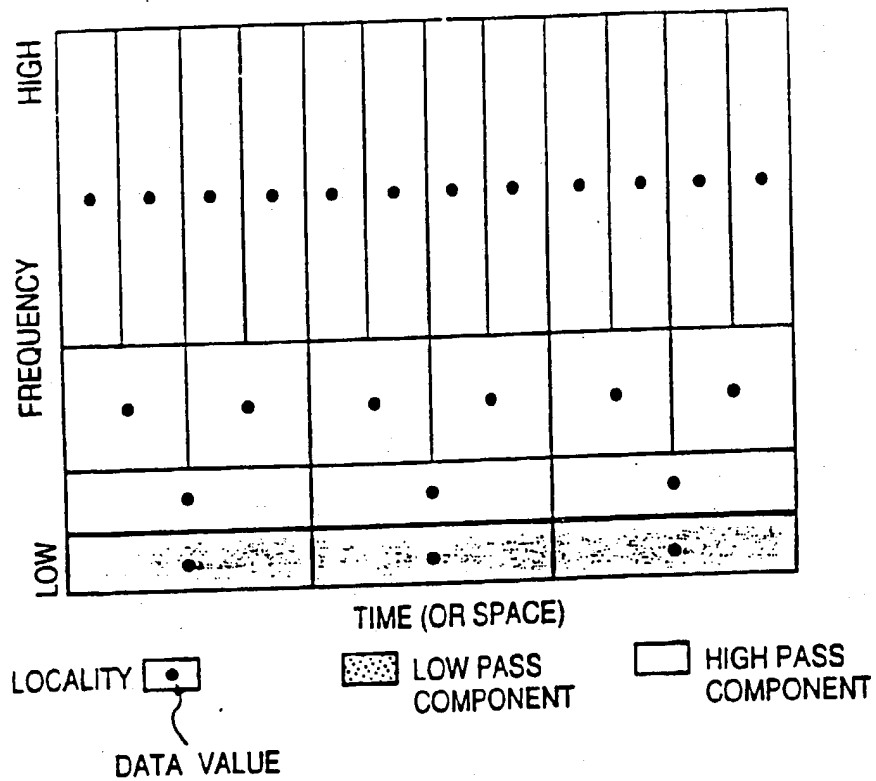


Fig. 7

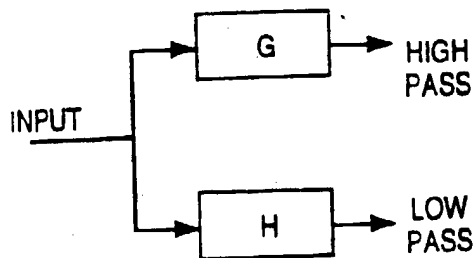


Fig. 8

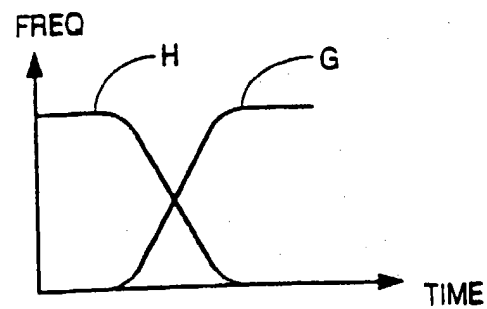


Fig. 9

4/24

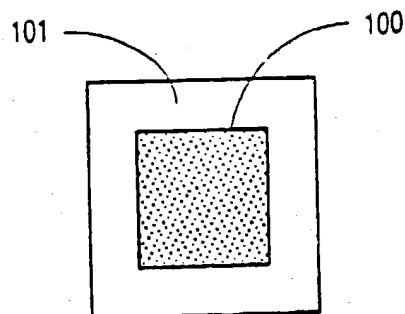


Fig. 10

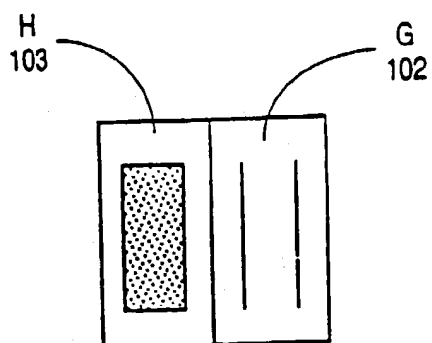


Fig. 11

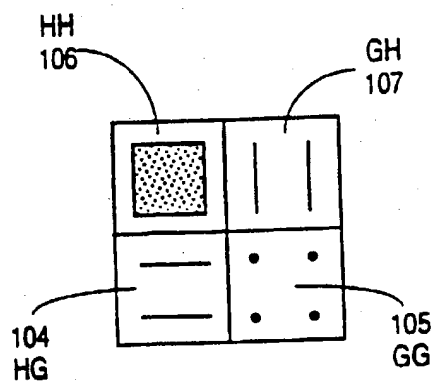


Fig. 14

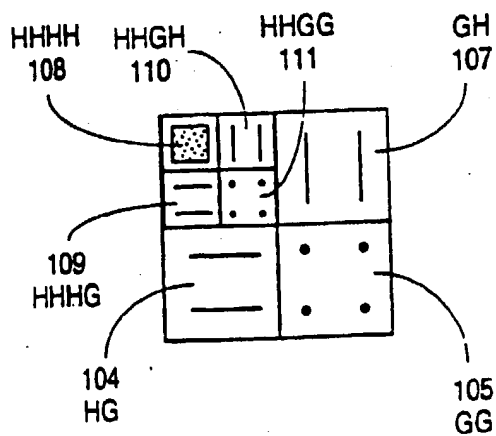


Fig. 15

5/24

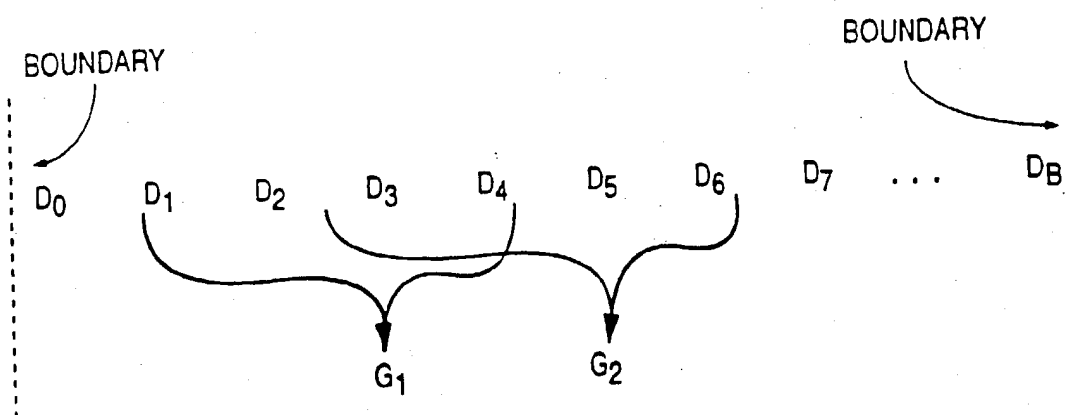


Fig. 12

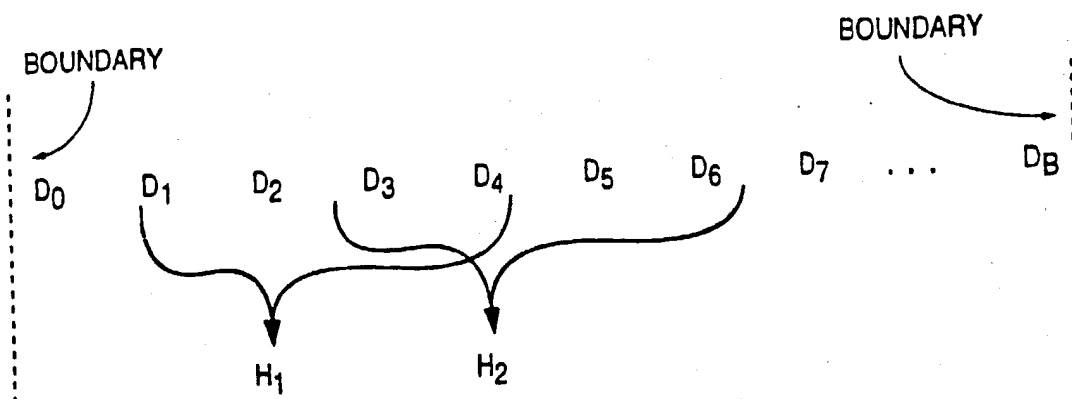


Fig. 13

6/24

COLUMN												
	0	1	2	3	4	5	6	7	8	9	A	B
0	D00	D01	D02	D03	D04	D05	D06	D07	D08	D09	D0A	D0B
1	D10	D11	D12	D13	D14	D15	D16	D17	D18	D19	D1A	D1B
2	D20	D21	D22	D23	D24	D25	D26	D27	D28	D29	D2A	D2B
3	D30	D31	D32	D33	D34	D35	D36	D37	D38	D39	D3A	D3B
4	D40	D41	D42	D43	D44	D45	D46	D47	D48	D49	D4A	D4B
5	D50	D51	D52	D53	D54	D55	D56	D57	D58	D59	D5A	D5B
6	D60	D61	D63	D63	D64	D65	D66	D67	D68	D69	D6A	D6B
7	D70	D71	D72	D73	D74	D75	D76	D77	D78	D79	D7A	D7B
8	D80	D81	D82	D83	D84	D85	D86	D87	D88	D89	D8A	D8B
9	D90	D91	D92	D93	D94	D95	D96	D97	D98	D99	D9A	D9B
A	DA0	DA1	DA2	DA3	DA4	DA5	DA6	DA7	DA8	DA9	DAA	DAB
B	DB0	DB1	DB2	DB3	DB4	DB5	DB6	DB7	DB8	DB9	DBA	DBB

R
O
W

Fig. 16

7/24

	COLUMN											
	0	1	2	3	4	5	6	7	8	9	A	B
0	HH00	GH00	HH01	GH01	HH02	GH02	HH03	GH03	HH04	GH04	HH05	GH05
1	HG00	GG00	HG01	GG01	HG02	GG02	HG03	GG03	HG04	GG04	HG05	GG05
2	HH10	GH10	HH11	GH11	HH12	GH12	HH13	GH13	HH14	GH14	HH15	GH15
3	HG10	GG10	HG11	GG11	HG12	GG12	HG13	GG13	HG14	GG14	HG15	GG15
4	HH20	GH20	HH21	GH21	HH22	GH22	HH23	GH23	HH24	GH24	HH25	GH25
5	HG20	GG20	HG21	GG21	HG22	GG22	HG23	GG23	HG24	GG24	HG25	GG25
6	HH30	GH30	HH31	GH31	HH32	GH32	HH33	GH33	HH34	GH34	HH35	GH35
7	HG30	GG30	HG31	GG31	HG32	GG32	HG33	GG33	HG34	GG34	HG35	GG35
8	HH40	GH40	HH41	GH41	HH42	GH42	HH43	GH43	HH44	GH44	HH45	GH45
9	HG40	GG40	HG41	GG41	HG42	GG42	HG43	GG43	HG44	GG44	HG45	GG45
A	HH50	GH50	HH51	GH51	HH52	GH52	HH53	GH53	HH54	GH54	HH55	GH55
B	HG50	GG50	HG51	GG51	HG52	GG52	HG53	GG53	HG54	GG54	HG55	GG55

R
O
W

Fig. 17

8/24

COLUMN												
	0	1	2	3	4	5	6	7	8	9	A	B
0	HHHH00 GH00	HHGH00 GH01	HHHH01 GH02	HHGH01 GH03	HHHH02 GH04	HHGH02 GH05						
1	HG00 GG00	HG01 GG01	HG02 GG02	HG03 GG03	HG04 GG04	HG05 GG05						
2	HHHG00 GH10	HHGG00 GH11	HHHG01 GH12	HHGG01 GH13	HHHG02 GH14	HHGG02 GH15						
3	HG10 GG10	HG11 GG11	HG12 GG12	HG13 GG13	HG14 GG14	HG15 GG15						
4	HHHH10 GH20	HHGH10 GH21	HHHH11 GH22	HHGH11 GH23	HHHH12 GH24	HHGH12 GH25						
5	HG20 GG20	HG21 GG21	HG22 GG22	HG23 GG23	HG24 GG24	HG25 GG25						
6	HHHG10 GH30	HHGG10 GH31	HHHG11 GH32	HHGG11 GH33	HHHG12 GH34	HHGG12 GH35						
7	HG30 GG30	HG31 GG31	HG32 GG32	HG33 GG33	HG34 GG34	HG35 GG35						
8	HHHH20 GH40	HHGH20 GH41	HHHH21 GH42	HHGH21 GH43	HHHH22 GH44	HHGH22 GH45						
9	HG40 GG40	HG41 GG41	HG42 GG42	HG43 GG43	HG44 GG44	HG45 GG45						
A	HHHG20 GH50	HHGG20 GH51	HHHG21 GH52	HHGG21 GH53	HHHG22 GH54	HHGG22 GH55						
B	HG50 GG50	HG51 GG51	HG52 GG52	HG53 GG53	HG54 GG54	HG55 GG55						

Fig. 18

9/24

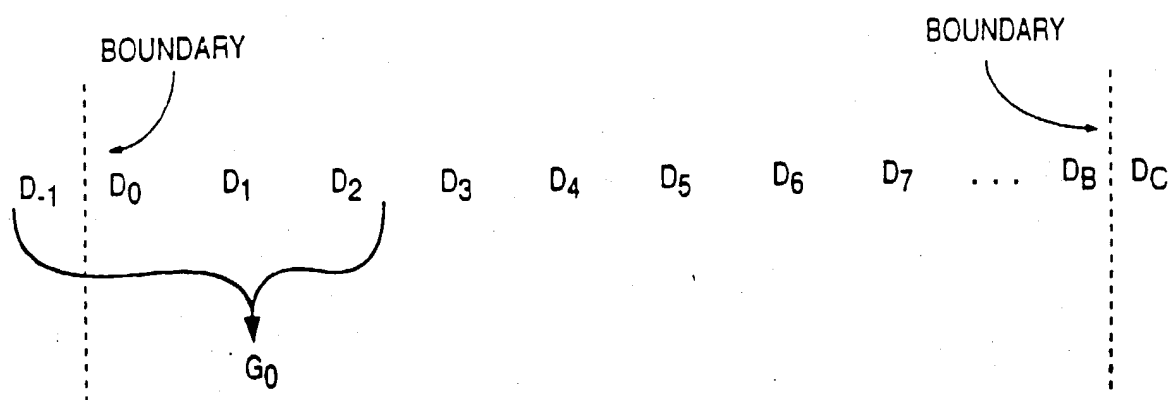


Fig. 19

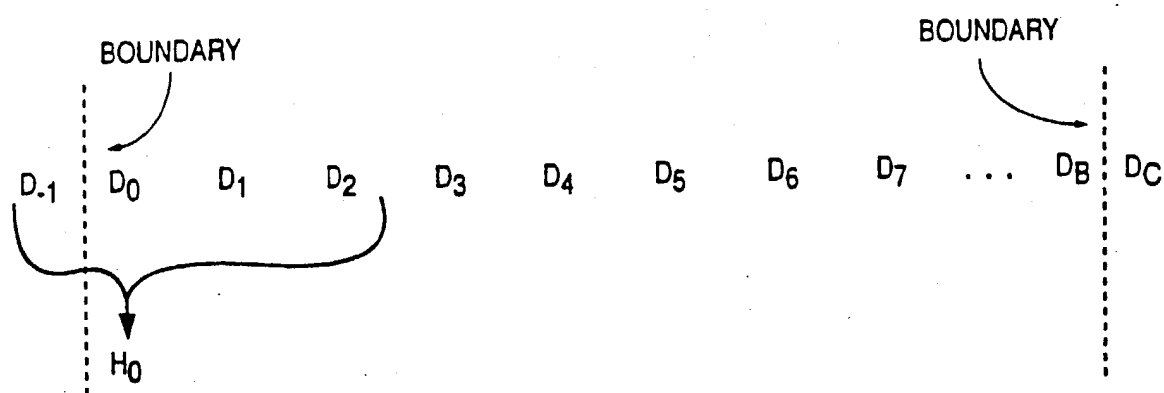


Fig. 20

10/24

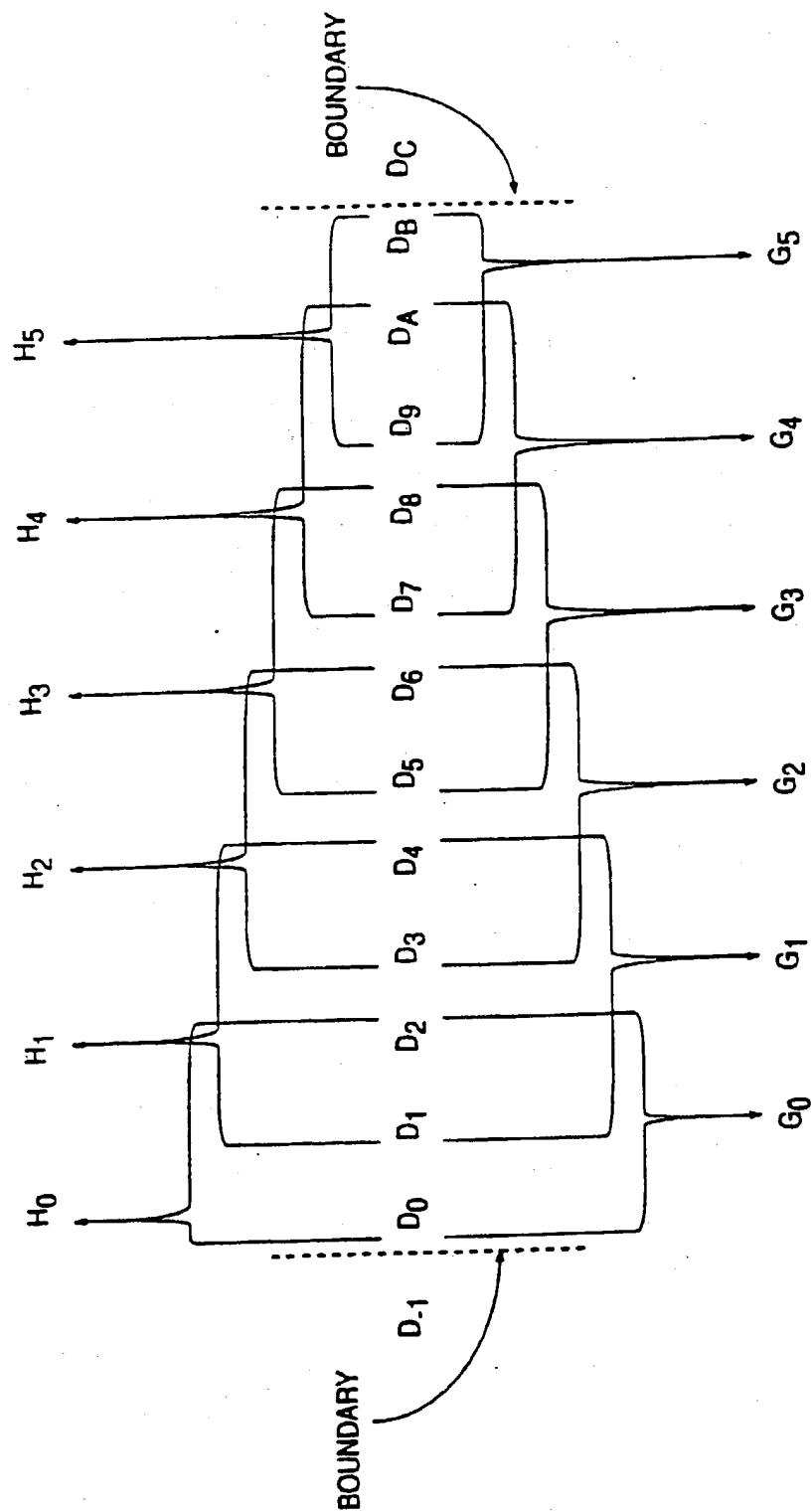


Fig. 21

11/24

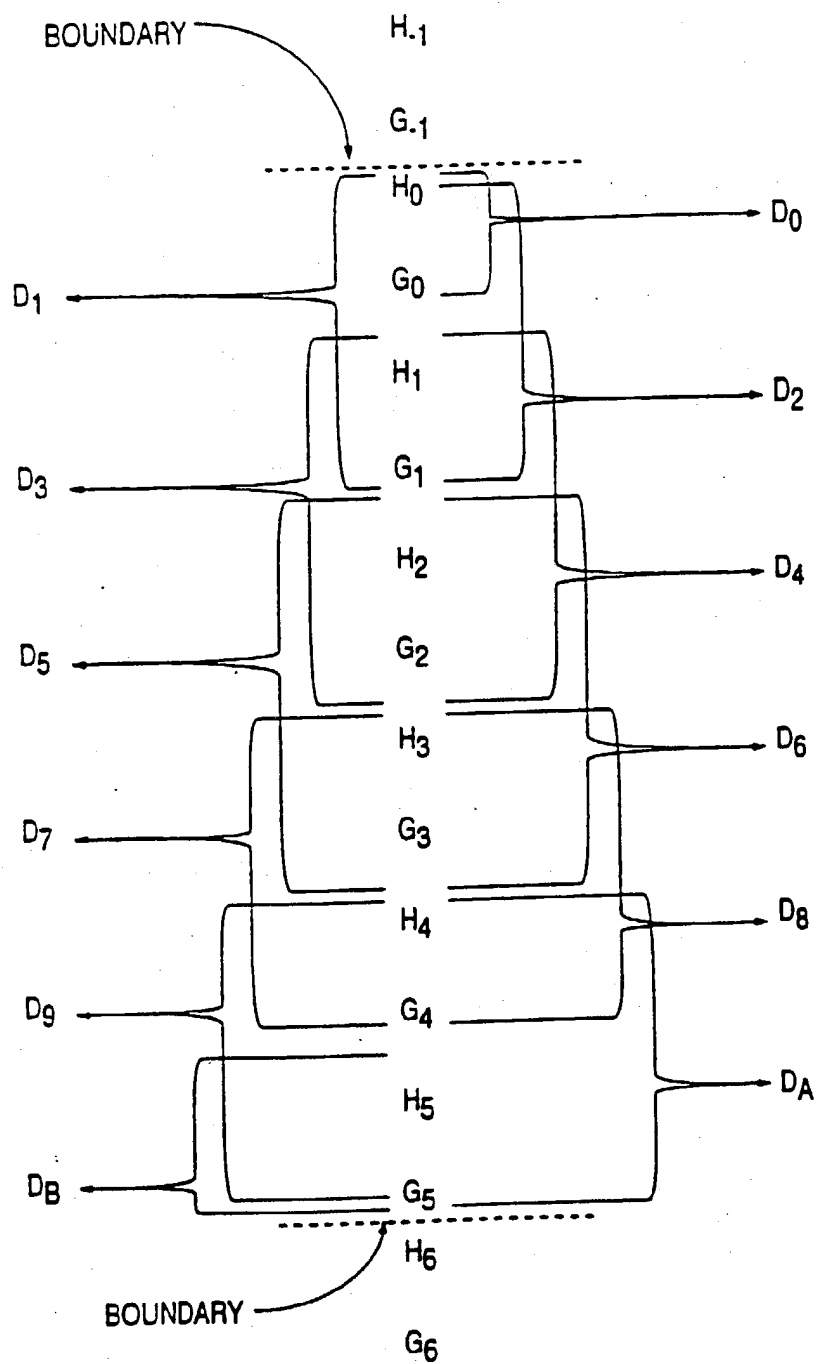


Fig. 22

12/24

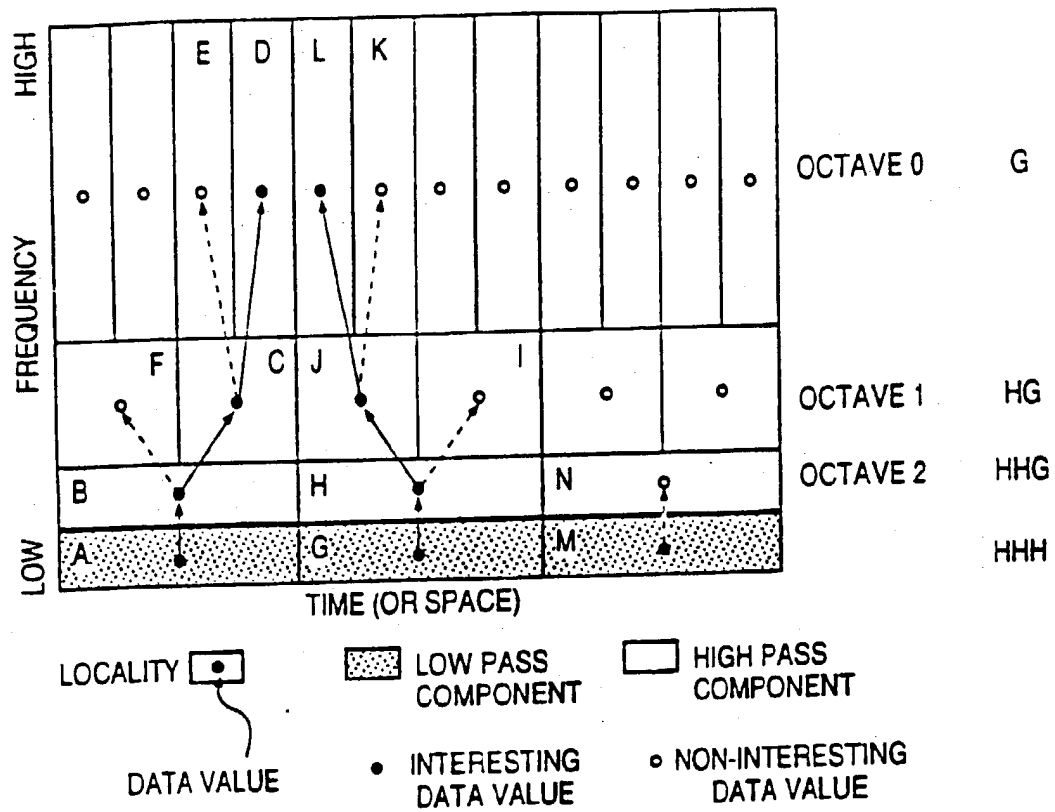


Fig. 23

13/24

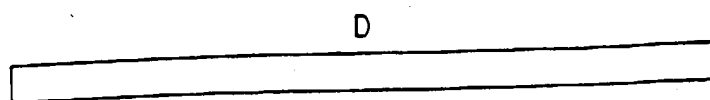


Fig. 24A

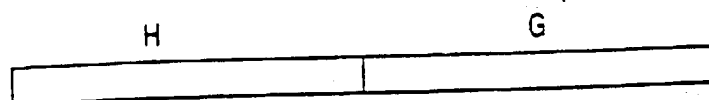


Fig. 24B

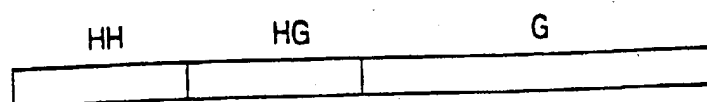


Fig. 24C

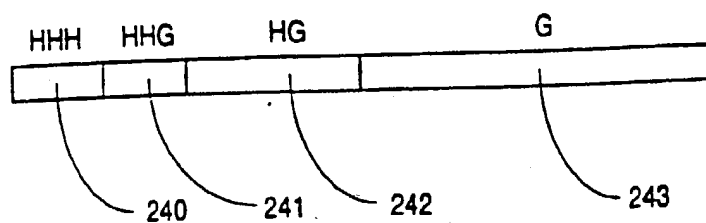


Fig. 24D

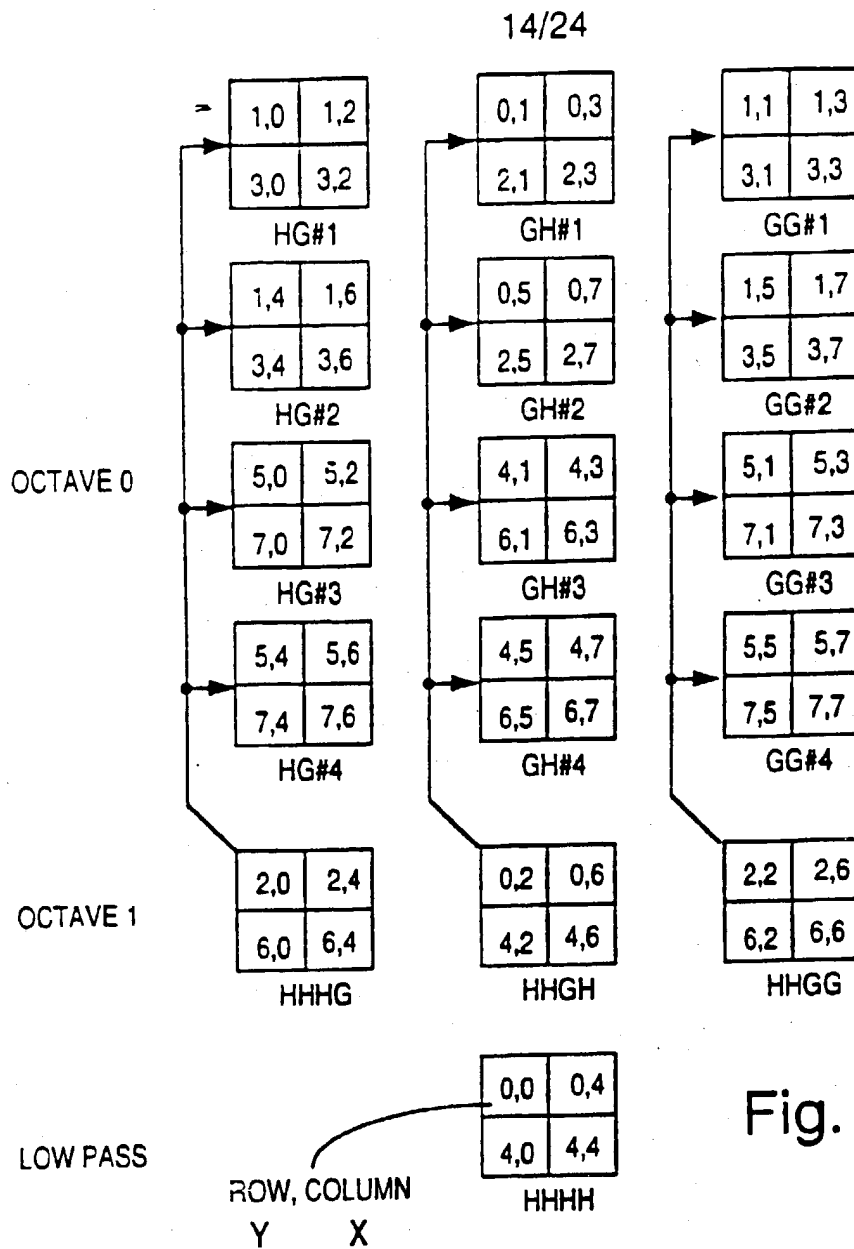
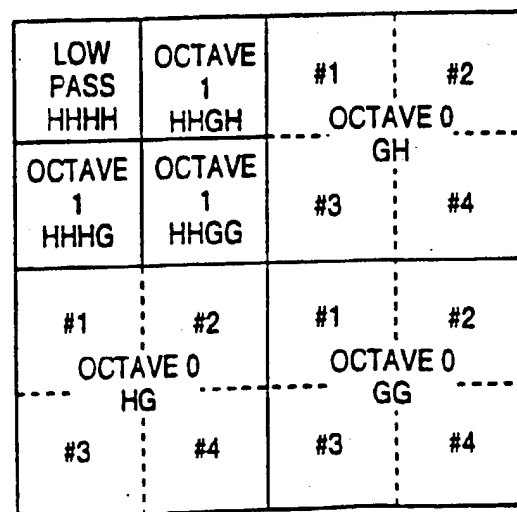


Fig. 25



PICTORIAL REPRESENTATION

Fig. 26

15/24

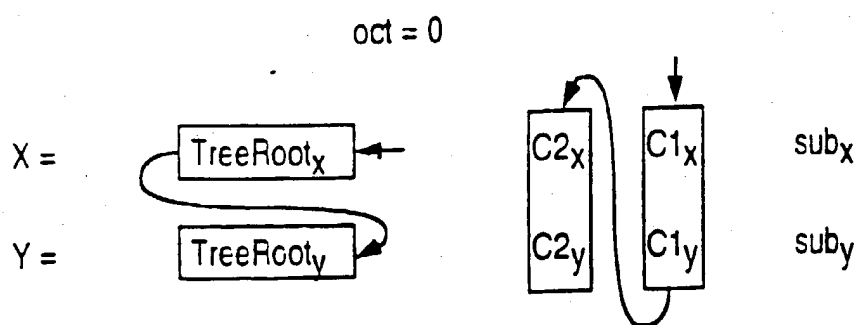


Fig. 27

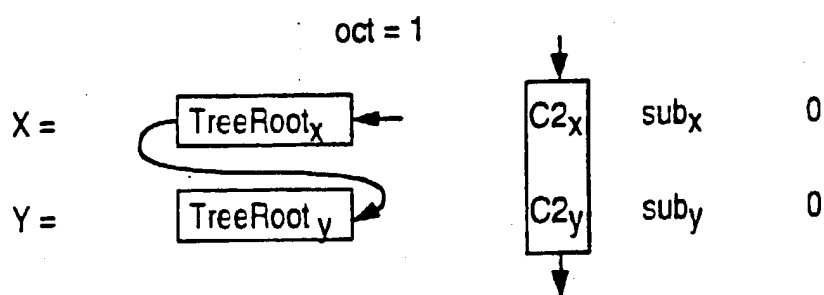


Fig. 28

sub-band		sub _x	sub _y
low pass	{ HH	0	0
	{ HG	0	1
high pass	{ GH	1	0
	{ GG	1	1

Fig. 29

16/24

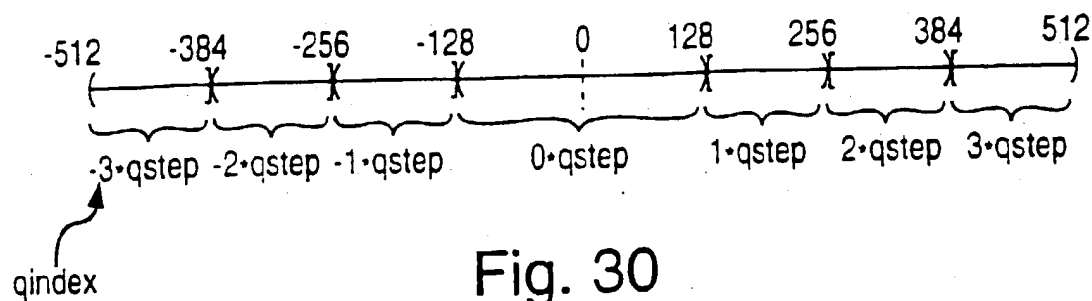


Fig. 30

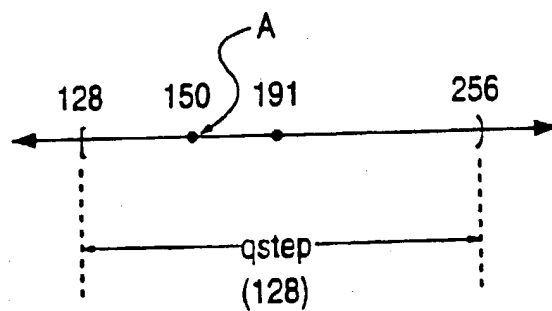


Fig. 31

17/24

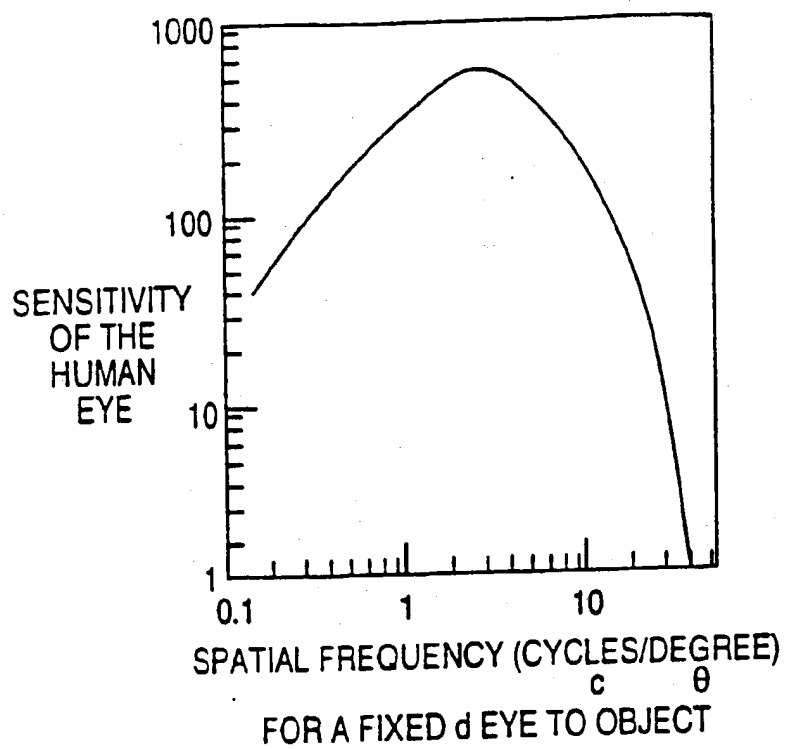


Fig. 32

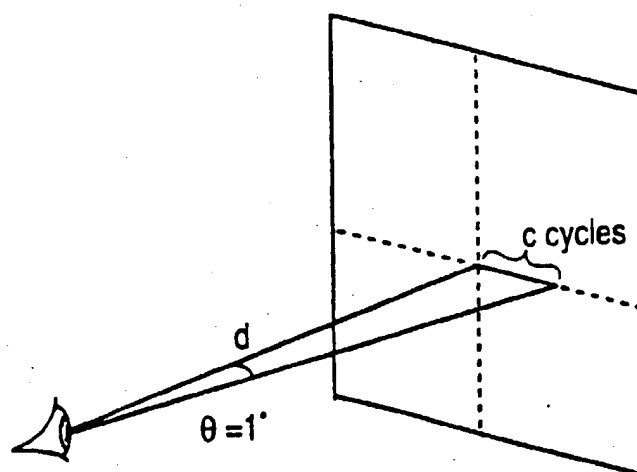


Fig. 33

18/24

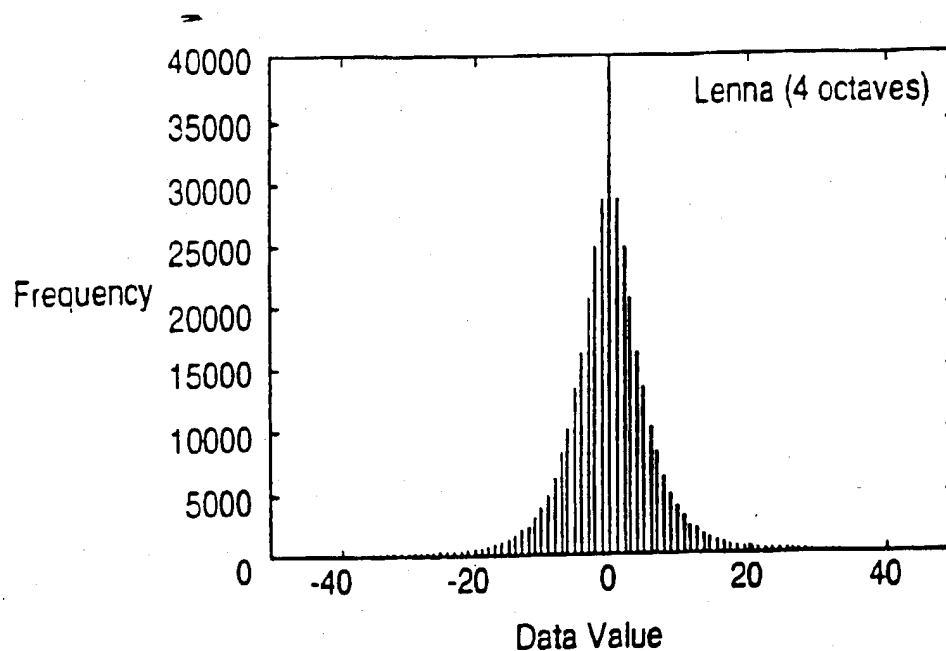


Fig. 34

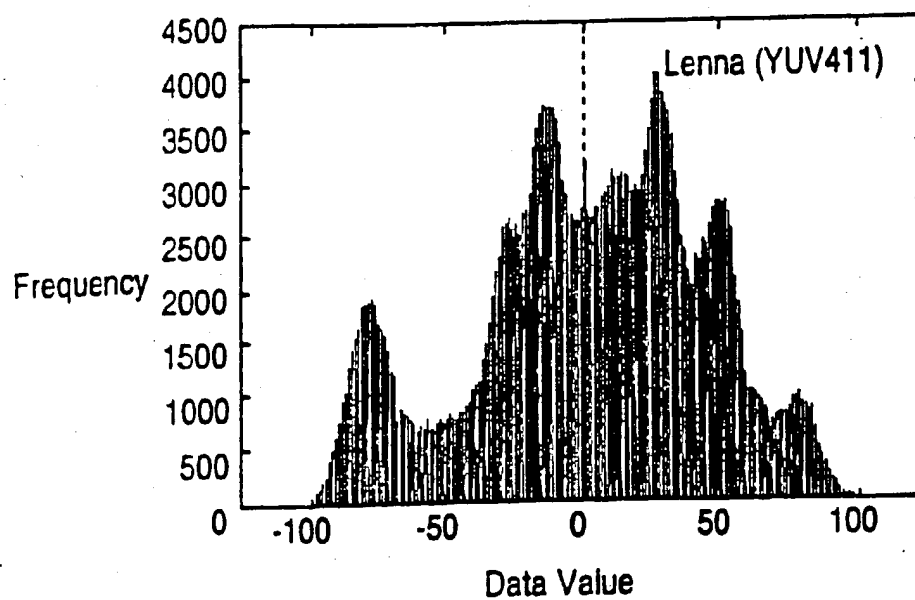


Fig. 35

19/24

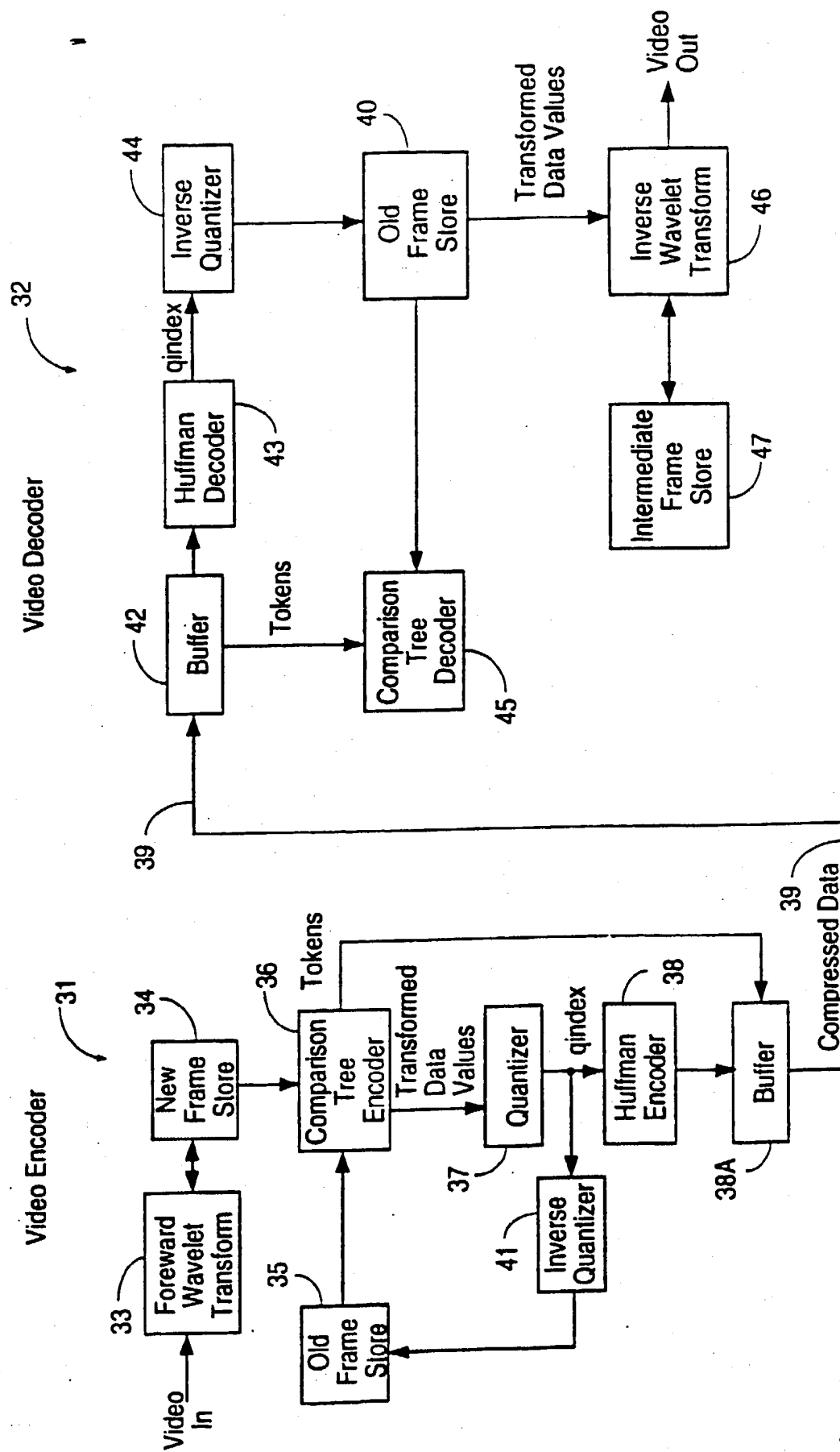


Fig. 36

20/24

MODES OF VIDEO ENCODER AND VIDEO DECODER

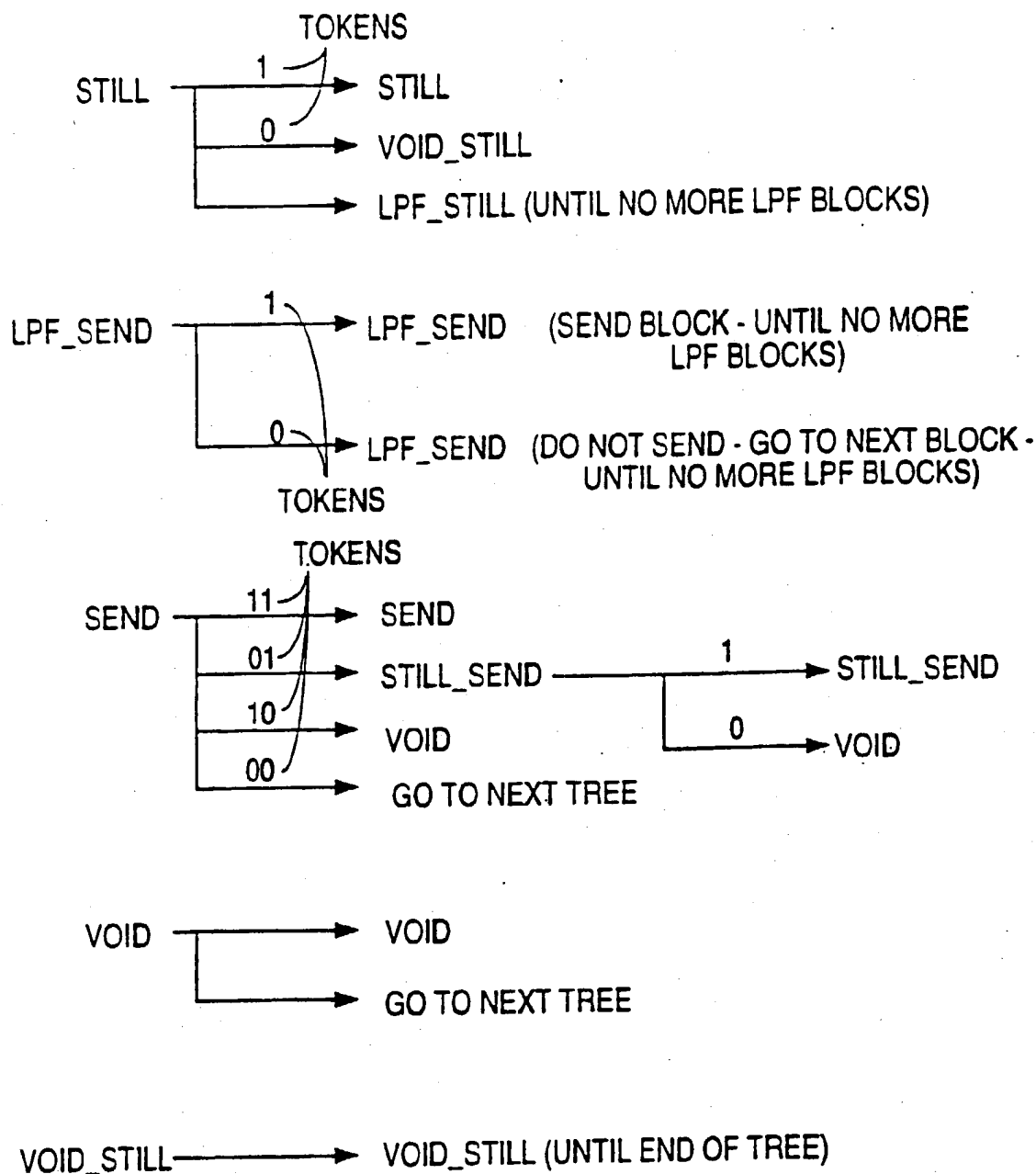


Fig. 37

21/24

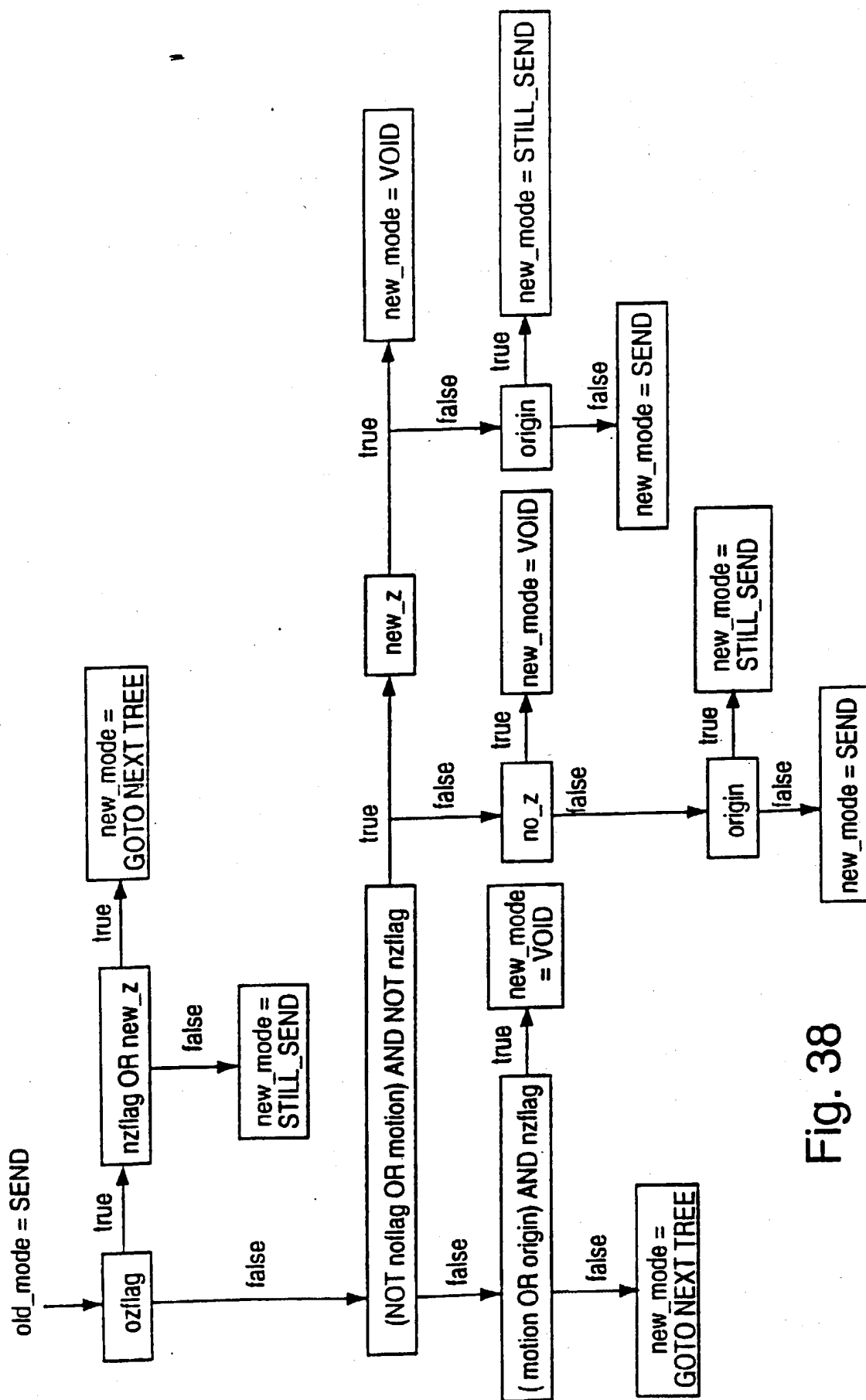


Fig. 38

22/24

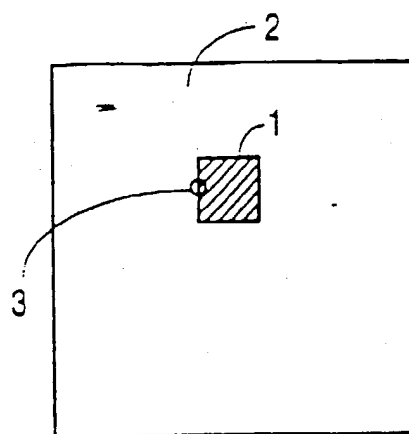


Fig. 39

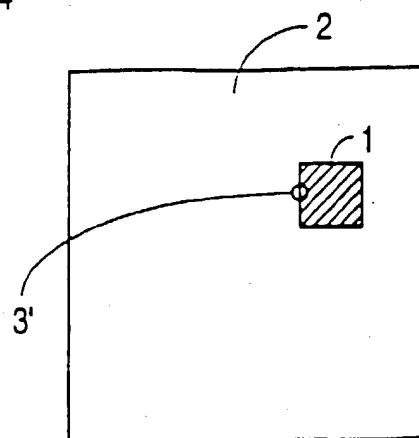


Fig. 40

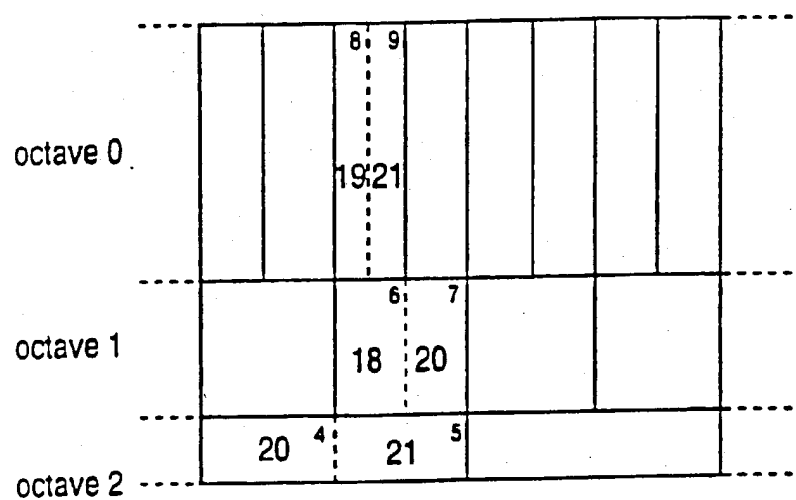


Fig. 41

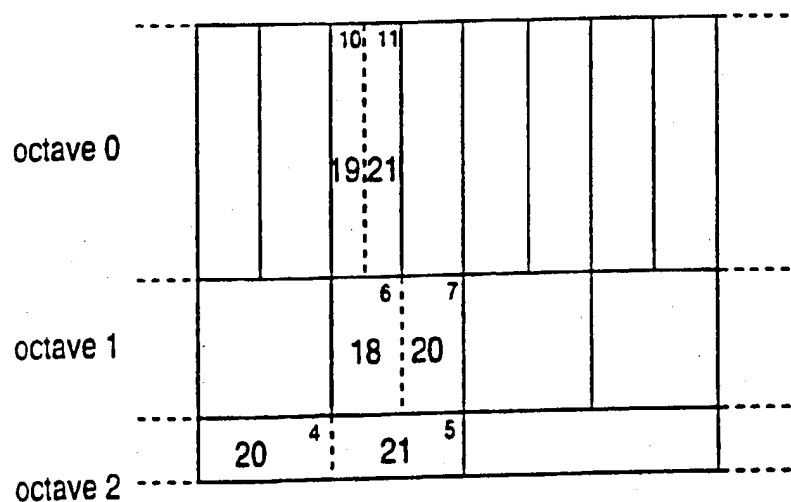


Fig. 42

23/24

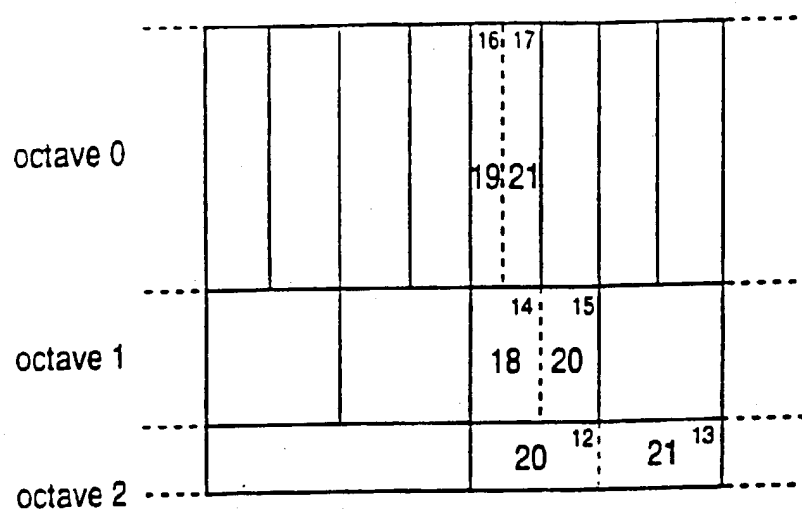


Fig. 43

24/24

VARIABLE - LENGTH TOKENS

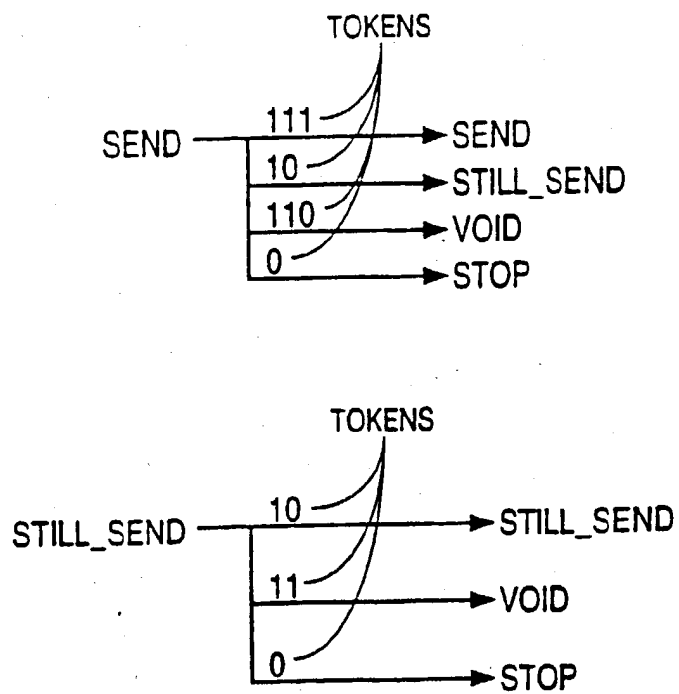


FIG. 44